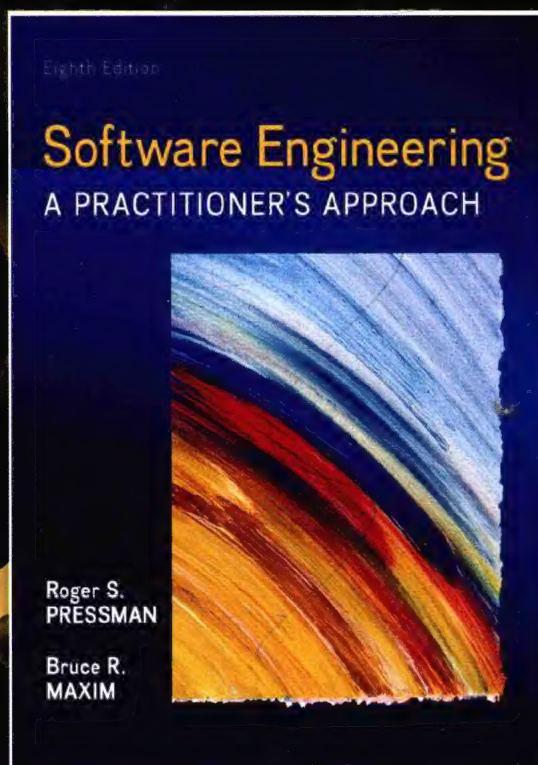


软件工程

实践者的研究方法

[美] 罗杰 S. 普莱斯曼 (Roger S. Pressman) 著
布鲁斯 R. 马克西姆 (Bruce R. Maxim)
郑人杰 马素霞 等译

Software Engineering
A Practitioner's Approach Eighth Edition



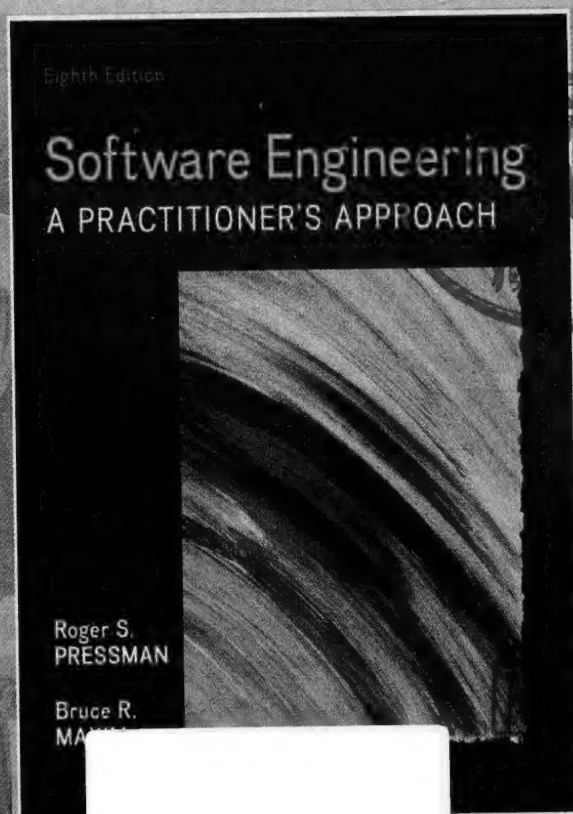
计 算 机 科 学 丛 书

软件工程

实践者的研究方法

[美] 罗杰 S. 普莱斯曼 (Roger S. Pressman) 著
布鲁斯 R. 马克西姆 (Bruce R. Maxim) 著
郑人杰 马素霞 等译

Software Engineering
A Practitioner's Approach Eighth Edition



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

软件工程: 实践者的研究方法 (原书第 8 版) / (美) 罗杰 S. 普莱斯曼 (Roger S. Pressman), (美) 布鲁斯 R. 马克西姆 (Bruce R. Maxim) 著; 郑人杰等译. —北京: 机械工业出版社, 2016.9

(计算机科学丛书)

书名原文: Software Engineering: A Practitioner's Approach, Eighth Edition

ISBN 978-7-111-54897-3

I. 软… II. ①罗… ②布… ③郑… III. 软件工程 IV. TP311.5

中国版本图书馆 CIP 数据核字 (2016) 第 226945 号

本书版权登记号: 图字: 01-2014-4760

Roger S. Pressman, Bruce R. Maxim: Software Engineering: A Practitioner's Approach, Eighth Edition (978-0-07-802212-8).

Copyright © 2015 by McGraw-Hill Education.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including without limitation photocopying, recording, taping, or any database, information or retrieval system, without the prior written permission of the publisher.

This authorized Chinese translation edition is jointly published by McGraw-Hill Education and China Machine Press. This edition is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Copyright © 2016 by McGraw-Hill Education and China Machine Press.

版权所有。未经出版人事先书面许可, 对本出版物的任何部分不得以任何方式或途径复制或传播, 包括但不限于复印、录制、录音, 或通过任何数据库、信息或可检索的系统。

本授权中文简体字翻译版由麦格劳-希尔(亚洲)教育出版公司和机械工业出版社合作出版。此版本经授权仅限在中华人民共和国境内(不包括香港、澳门特别行政区及台湾地区)销售。

版权 © 2016 由麦格劳-希尔(亚洲)教育出版公司与机械工业出版社所有。

本书封面贴有 McGraw-Hill Education 公司防伪标签, 无标签者不得销售。

本书自第 1 版出版至今, 30 多年来在软件工程界产生了巨大而深远的影响。第 8 版不仅加入了移动 App 项目等与时俱进的内容, 而且调整了篇章结构, 更利于教师针对不同课程进行选择。同时, 第 8 版继承了之前版本的风格与优势, 全面且系统地讲解软件过程、建模、质量管理、项目管理等基础知识, 涵盖相关概念、原则、方法和工具, 并且提供丰富的辅助阅读资源和网络资源, 指导有兴趣的读者进行更深入的学习和研究。

本书适合作为软件工程相关专业高年级本科生和研究生教材, 也可供软件专业技术人员和管理人员阅读参考。

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 曲 熠

责任校对: 殷 虹

印 刷: 北京市荣盛彩色印刷有限公司

版 次: 2016 年 11 月第 1 版第 1 次印刷

开 本: 185mm×260mm 1/16

印 张: 43

书 号: ISBN 978-7-111-54897-3

定 价: 99.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光/邹晓东

文艺复兴以来,源远流长的科学精神和逐步形成的学术规范,使西方国家在自然科学的各个领域中取得了垄断性的优势;也正是这样的优势,使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中,美国的产业界与教育界越来越紧密地结合,计算机科学中的许多泰山北斗同时身处科研和教学的最前线,由此而产生的经典科学著作,不仅擘划了研究的范畴,还揭示了学术的源变,既遵循学术规范,又自有学者个性,其价值并不会因年月的流逝而减退。

近年,在全球信息化大潮的推动下,我国的计算机产业发展迅猛,对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇,也是挑战;而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下,美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此,引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用,也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始,我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力,我们与Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage等世界著名出版公司建立了良好的合作关系,从他们现有的数百种教材中甄选出Andrew S. Tanenbaum, Bjarne Stroustrup, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson等大师名家的一批经典作品,以“计算机科学丛书”为总称出版,供读者学习、研究及珍藏。大理石纹理的封面,也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助,国内的专家不仅提供了中肯的选题指导,还不辞劳苦地担任了翻译和审校的工作;而原书的作者也相当关注其作品在中国的传播,有的还专门为其书的中译本作序。迄今,“计算机科学丛书”已经出版了近百个品种,这些书籍在读者中树立了良好的口碑,并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑,这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化,教育界对国外计算机教材的需求和应用都将步入一个新的阶段,我们的目标是尽善尽美,而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正,我们的联系方式如下:

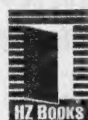
华章网站: www.hzbook.com

电子邮件: hzjsj@hzbook.com

联系电话: (010) 88379604

联系地址: 北京市西城区百万庄南街1号

邮政编码: 100037



华章教育

华章科技图书出版中心

译者序

Software Engineering: A Practitioner's Approach, Eighth Edition

本书是国际知名软件工程专家罗杰 S. 普莱斯曼 (Roger S. Pressman) 的最新著作。自 35 年前第 1 版问世以来, 这本书在软件工程界产生了巨大而深远的影响。其权威性是无可置疑的, 在培养软件工程专业人才方面所起的作用也是显而易见的。在这一版中, 新加入的布鲁斯 (Bruce) 作为第二作者参与了本书的编写工作。

我自 20 世纪 80 年代中期开始从事高校软件工程方面的教学与科研工作, 多年来, 这本书的各个版本一直是我的重要参考书, 它给了我许多启发和帮助, 我也曾多次向许多业界好友和学生推荐此书。

如今基于计算机的系统已经广泛而深入地渗透到经济、国防和人们日常生活的各个领域, 特别是在互联网的推动下, 不仅许多行业得以改进和更新, 而且产生了一批新的行业, 展现了全新的业态。我们必须意识到, 在计算机不断向社会的深度和广度层面发展的过程中, 软件始终处在系统的核心地位, 起着中枢和灵魂的作用, 而且这种作用正日益突出。因此, 如何为现代化系统配备合格和优良的软件也就更加受到人们的广泛关注。

本书系统地论证了软件工程领域的基本知识和最新研究成果, 包括新的概念、原则、技术、方法和工具。同时书中还为读者提供了进一步学习和研究的线索, 包括许多可供利用的网上资料和信息。与第 7 版相比, 本版继承了一些优点, 同时也做了一些改动、扩充和更新。

本书特点

1. 全书内容分为五个部分, 共 39 章, 还包括三个附录。五个部分的内容分别为软件过程、建模、质量管理、管理软件项目以及软件工程高级课题。
2. 本书继承了前一版的特色, 突出了软件质量管理的内容, 同时也加强了软件过程部分。此外, 增加的内容还包括: 软件工程项目对人员的要求; 近年来软件产业发展中出现的新课题——移动 App 和软件系统安全性。
3. 仍然在各章的开头给出“要点浏览”(包括概念、人员、重要性、步骤、工作产品和质量保证措施) 以及“关键概念”(全章内容的关键词)。
4. 仍然在各章的末尾给出“小结”“习题与思考题”以及“扩展阅读与信息资源”, 这些都非常适合有兴趣、有需要的读者沿着所提供的线索开展进一步的学习和研究。
5. 仍然保留了本书历次版本在各章中为读者提供的多种形式的辅助阅读信息, 可以说这是本书的一个突出特点。这些信息从形式上分为两类: 一类是采用通栏形式的说明框, 包括要点浏览、信息栏、软件工具和 SafeHome 对话框等; 另一类是居于页面右侧的说明框, 包括关键概念、引述、建议、关键点、提问和网络资源。这些说明框非常有益于读者理解和进一步探索相关内容。

读者对象

本书仍然面向三类读者, 即高校学生 (特别是研究生)、教师和专业软件技术人员。总

体而言,本书适合作为高校计算机或信息技术相关专业的教学用书,特别适合为软件工程课程提供教学服务。

对于采用本书作为教材的教师,在此提供几点建议:

1. 由于学时所限,不可能将本书的全部内容纳入教学,因此从中选取适合的部分是必然的。可以对软件管理部分做一些压缩,但即使如此,我认为也不可把有关管理的内容完全删除。
2. 近年来软件工程领域出现了许多新技术和新方法,作者已将其及时引入书中。但请教师注意,对于初学者来说,牢固地掌握基本概念、基本技能和方法仍然十分重要。

译者说明

参与本书翻译工作的译者以华北电力大学和清华大学的教师为主,也有少数软件企业和中国软件行业协会的研究人员。他们是:马素霞(第1~4章、22~25章、28章及附录1~3)、宋兰(第7~11章及29章)、石敏(第12、13章)、周长玉(第14、15章)、吴爽(第16~18章)、韩新启(第19~21章)、王海青(第30章及35~39章)、王素琴(第31~34章)。此外,刘瑾完成了第5、6章的翻译工作。我负责第26、27章以及前言和作者简介部分。在翻译过程中,我们得到了华北电力大学控制与计算机工程学院洪海、熊里、赵敏、李树超、高晶晶、吕骁同学的帮助,在此对他们的辛勤劳动表示感谢。我对全部译稿、马素霞教授对大部分译稿做了仔细审核与修改,并更正了原书中个别的错漏之处。

本书英文版有900多页,翻译工作量巨大,而译者均有繁重的本职工作,时间并不宽松,因此译文中难免有不当之处,敬请读者见谅并不吝指正。

总之,这是一本非常优秀的软件工程读物,本人十分高兴地向国内读者推荐。我们相信,认真阅读它,定会使你获益匪浅。

郑人杰

2016年7月

如果有这样一款计算机软件，它能满足用户的需求，能在相当长的时间内无故障地运行，修改起来轻松便捷，使用起来更是得心应手，那么，这款软件必定是成功的，它切实改善了我们的生活。但是，如果有这样一款软件，它令用户失望，错误频出，修改起来困难重重，使用起来更是举步维艰，那么，这必定是一款失败的软件，它使我们的生活一团糟。谁都希望开发出优秀的软件，为我们的生活带来便利，而不是把自己陷入失败的深渊。要想使软件获得成功，在设计和构建软件时就需要有规范，需要采用工程化的方法。

自本书第1版问世以来的近35年中，软件工程已经从少数倡导者提出的一些朦胧概念发展成为一门正规的工程学科，已被公认为是一个值得深入研究、认真学习和热烈讨论的课题。在整个行业中，软件工程师已经代替程序员成为人们优先选择的工作岗位，软件过程模型、软件工程方法和软件工具都已在全行业的所有环节成功采用。

尽管管理人员和一线专业人员都承认需要有更为规范的软件方法，但他们却始终在争论应该采用什么样的规范。有许多个人和公司至今仍在杂乱无章地开发着自己的软件，甚至即使他们正在开发的系统要服务于当今最为先进的技术，状况也仍是如此。许多专业人员和学生并不了解现代方法，这导致他们所开发的软件质量很差，因而造成了严重的后果。此外，有关软件工程方法真实本质的争论一直持续进行着。软件工程的地位问题已成为一门对比研究课题。人们对软件工程的態度已经有所改善，研究工作已取得了进展，不过要成为一门完全成熟的学科，我们还有大量的工作要做。

我们希望本书能够成为引导读者进入正在走向成熟的软件工程学科的入门读物，和以前的7个版本一样，第8版对学生和专业人员同样具有很强的吸引力。它既是软件专业人员的工作指南，也是高年级本科生和一年级研究生的综合性参考书。

第8版中包含了许多新的内容，它绝不只是前一版的简单更新。这一版不仅对内容做了适当的修改，而且调整了全书的结构，以改进教学顺序；同时更加强调一些新的和重要的软件工程过程和软件工程实践知识。此外，本书进一步加强了“支持系统”，为学生、教师和专业人员提供了更为丰富的知识资源。读者可访问专门为本书建立的网站（www.mhhe.com/pressman）查阅这些信息。

篇章结构

本书共39章，分为五个部分。这种划分有利于那些无法在一个学期内讲完全书内容的教师灵活安排教学。

第一部分“软件过程”给出了有关软件过程的各种不同观点，讨论了所有重要的过程模型，还涉及惯用过程和敏捷过程在指导思想上的分歧。第二部分“建模”给出了分析方法和设计方法，重点讲解面向对象方法和UML建模，同时也介绍了基于模式的设计以及基于WebApp和移动App的设计。第三部分“质量管理”介绍了有关质量管理的概念、规程、技术和方法，使得软件团队能够很好地评估软件质量，评审软件工程项目产品，实施软件质量保证规程，并正确地运用有效的测试策略和战术。此外，这一部分还讨论了形式化建模和验

证方法。第四部分“管理软件项目”介绍了与计划、管理和控制软件开发项目的人员有关的问题。第五部分“软件工程高级课题”讨论了软件过程改进和软件工程的发展趋势。

第8版沿用了前面几个版本的做法，在各章中都提供了大量的辅助阅读信息，包括一个虚拟软件团队在工作中遇到困难时展开的对话，还包括对各章相关知识给出的补充方法和工具。

第8版中五个部分的划分有利于教师根据学时和教学要求安排课堂内容。在一个学期内可以安排一个部分的内容，也可以安排多个部分的内容。软件工程概论课程可以从五个部分中选择若干章作为教材。侧重分析和设计的软件工程课程可以从第一部分和第二部分中选取素材。面向测试的软件工程课程则可以从第一部分和第三部分中选取素材，还应加上第二部分中的一些内容。侧重管理的课程应突出第一部分和第四部分的内容。我们用上述方式组织第8版的内容，意在给教师提供多种教学安排的选择。但无论如何选择这些内容，都可以从“支持系统”中获得补充资源。

相关资源^①

学生资源

本书为学生提供的各种学习资料包括：在线学习中心提供的各章学习指南，实践测验，题解以及多种在线资源（软件工程检查单、一套正在演化的微型工具、综合案例研究和和工作产品模板等）。此外，1000多种网络参考文献可供学生更深入地探究软件工程问题，还有500多篇可下载的参考文献，这些都为读者提供了关于高级软件工程课题的更为详尽的信息。

教师资源

本书为教师提供的各种教学资料包括：在线（也可下载）教师指南，由700多个PPT组成的教辅资源和试题库。当然，学生资源（如微型工具、网络参考文献及可下载参考文献）和专业人员资源也可供教师使用。

在本书的教师指南中，我们为各种类型的软件工程课程提出了建议，介绍了与课程配合开展的软件项目、部分问题的题解和许多有用的教学辅助工具。

专业人员资源

本书为产业界专业人员（也包括在校学生）提供的各种资料包括：软件工程文档和其他工作产品的大纲和模板，一套有用的软件工程检查单，软件工程工具目录，综合性网络资料以及用于软件工程过程具体任务划分的“通用过程模型”。

由于有了在线支持系统的配合，使得本书既有内容上的深度，又有一定的灵活性，这些优势是传统的教科书所无法比拟的。

布鲁斯·马克西姆（Bruce Maxim）为这一版贡献了新的力量，他不仅具备丰富的软件工程知识，还带来了新的内容和洞见，这些对于读者来说将是十分宝贵的。

① 关于本书资源，请访问 www.mhhe.com/pressman 查看，有需要的读者可向麦格劳·希尔教育出版公司北京代表处申请，电话：8008101936/010-62790299-108，电子邮件：instructorchina@mcgraw-hill.com。——编辑注

致谢

我们要特别感谢渥太华大学的 Tim Lethbridge, 他帮助我们开发了 UML 和 OCL 的案例, 以及配合本书内容的其他案例研究。Colby 学院的 Dale Skrien 开发了附录 1 的 UML 教辅资源。他们的帮助和意见都是十分宝贵的。此外也感谢高级软件工程师 Austin Krauss, 他提供了关于视频游戏产业软件开发的宝贵意见。同时, 要对为第 8 版评审做出贡献的几位教授表示感谢, 他们是佛罗里达大学的 Manuel E. Bermudez、堪萨斯州立大学的 Scott DeLoach、密歇根州立大学的 Alex Liu 和犹他州立大学的 Dean Mathias。正是他们的详尽而真诚的评审意见帮助我们, 使得本书更加完善。

特别感谢

十分高兴有机会与罗杰合作, 参与本书第 8 版的撰写工作。在此期间我的儿子 Benjamin 推出了他的第一款移动 App, 我的女儿 Katherine 开始了她的室内设计生涯。我十分高兴地看到他们已经长大成人。同时非常感谢妻子 Norma, 她热情地支持我, 使我能够将所有空闲时间都投入本书的写作。

布鲁斯 R. 马克西姆 (Bruce R. Maxim)

随着本书各版本的不断推出, 我的两个儿子 Mathew 和 Michael 也逐渐从小男孩成长为男子汉。他们在生活中的成熟、品格和成功鼓舞着我, 没有什么比这更让我自豪了。他们现在也已经有了自己的孩子——Maya 和 Lily, 这两个女孩已经是移动计算时代新智能设备方面的奇才。最后要感谢妻子 Barbara, 她宽容我花费如此多的时间在办公室工作, 并且还鼓励我继续写作本书的下一个版本。

罗杰 S. 普莱斯曼 (Roger S. Pressman)

罗杰 S. 普莱斯曼 (Roger S. Pressman)

普莱斯曼是软件工程领域国际知名的顾问和作家。40 多年来，他作为工程师、经理人、教授、作家、咨询师和企业家始终奋战在这一领域。

普莱斯曼博士现任一家咨询公司 (R. S. Pressman & Associates, Inc.) 的总裁，该公司致力于协助企业建立有效的软件工程实践。这些年来，他已经开发了一套用于改进软件工程实践的技术和工具。他还是一家创业公司 (Teslaccessories, LLC) 的创始人，这家制造公司专门为特斯拉 Model S 系列电动车生产定制产品。

普莱斯曼博士是 9 本书的作者，其中包括两本小说。他还写了许多技术和管理方面的文章。他曾任《IEEE Software》和《The Cutter IT Journal》等行业杂志的编委，以及《IEEE Software》杂志“Manager”专栏的编辑。

普莱斯曼博士还是著名的演讲家，曾在许多重要的行业会议上做主题演讲，在国际软件工程会议和一些行业会议上做辅导讲座，并且一直是 ACM (美国计算机协会)、IEEE (美国电气与电子工程师协会) 以及 Tau Beta Pi、Phi Kappa Phi、Eta Kappa Nu 和 Pi Tau Sigma 等组织的成员。

布鲁斯 R. 马克西姆 (Bruce R. Maxim)

马克西姆博士 30 多年来曾任软件工程师、项目经理、教授、作家和咨询师。他的研究兴趣涉及软件工程、人机交互、游戏设计、社交媒体、人工智能以及计算机科学教育等领域。

马克西姆博士现任密歇根大学迪尔伯恩分校计算机与信息科学系副教授，他曾为该校工程与计算机科学学院建立游戏实验室。他曾经发表多篇有关计算机算法动画、游戏开发以及工程教育方面的论文。他还是畅销的计算机科学导论课本的作者之一。在密歇根大学工作期间，马克西姆博士曾监管了几百个产业界软件开发项目。

马克西姆博士的专业经验包括在医学院管理研究信息系统，为某医学校区指导计算教学，并承担统计程序员的工作。他还曾担任某游戏开发公司的首席技术官。

马克西姆博士是若干著名教学奖以及某著名社团组织服务奖的获得者。他还是 Sigma Xi、Upsilon Pi Epsilon、Pi Mu Epsilon、ACM、IEEE、美国工程教育协会、女工程师协会以及国际游戏开发者联盟等社会组织的成员。

出版者的话
译者序
前言
作者简介

第 1 章 软件的本质	1
1.1 软件的本质	3
1.1.1 定义软件	3
1.1.2 软件应用领域	4
1.1.3 遗留软件	5
1.2 软件的变更本质	6
1.2.1 WebApp	6
1.2.2 移动 App	7
1.2.3 云计算	7
1.2.4 产品线软件	8
1.3 小结	8
习题与思考题	8
扩展阅读与信息资源	9
第 2 章 软件工程	10
2.1 定义软件工程学科	11
2.2 软件过程	11
2.2.1 过程框架	12
2.2.2 普适性活动	12
2.2.3 过程的适应性调整	13
2.3 软件工程实践	13
2.3.1 实践的精髓	14
2.3.2 通用原则	14
2.4 软件开发神话	16
2.5 这一切是如何开始的	18
2.6 小结	19
习题与思考题	19
扩展阅读与信息资源	19

第一部分 软件过程

第 3 章 软件过程结构	22
3.1 通过程模型	23
3.2 定义框架活动	24
3.3 明确任务集	24
3.4 过程模式	25
3.5 过程评估与改进	27
3.6 小结	28
习题与思考题	28
扩展阅读与信息资源	28
第 4 章 过程模型	29
4.1 惯用过程模型	30
4.1.1 瀑布模型	30
4.1.2 增量过程模型	32
4.1.3 演化过程模型	32
4.1.4 并发模型	36
4.1.5 演化过程的最终评述	37
4.2 专用过程模型	38
4.2.1 基于构件的开发	38
4.2.2 形式化方法模型	39
4.2.3 面向方面的软件开发	39
4.3 统一过程	40
4.3.1 统一过程的简史	41
4.3.2 统一过程的阶段	41
4.4 个人过程模型和团队过程模型	42
4.4.1 个人软件过程	42
4.4.2 团队软件过程	43
4.5 过程技术	44
4.6 产品和过程	45
4.7 小结	46
习题与思考题	46
扩展阅读与信息资源	47

第5章 敏捷开发	48
5.1 什么是敏捷	49
5.2 敏捷及变更成本	50
5.3 什么是敏捷过程	50
5.3.1 敏捷原则	51
5.3.2 敏捷开发战略	52
5.4 极限编程	52
5.4.1 极限编程过程	52
5.4.2 工业极限编程	54
5.5 其他敏捷过程模型	56
5.5.1 Scrum	56
5.5.2 动态系统开发方法	57
5.5.3 敏捷建模	58
5.5.4 敏捷统一过程	59
5.6 敏捷过程工具集	60
5.7 小结	61
习题与思考题	61
扩展阅读与信息资源	62
第6章 软件工程的人员方面	64
6.1 软件工程师的特质	64
6.2 软件工程心理学	65
6.3 软件团队	66
6.4 团队结构	67
6.5 敏捷团队	68
6.5.1 通用敏捷团队	68
6.5.2 XP 团队	69
6.6 社交媒体的影响	70
6.7 软件工程中云的应用	71
6.8 协作工具	71
6.9 全球化团队	72
6.10 小结	73
习题与思考题	73
扩展阅读与信息资源	74

第二部分 建模

第7章 指导实践的原则	76
7.1 软件工程知识	77
7.2 核心原则	77
7.2.1 指导过程的原则	78
7.2.2 指导实践的原则	78
7.3 指导每个框架活动的原则	80
7.3.1 沟通原则	80
7.3.2 策划原则	81
7.3.3 建模原则	83
7.3.4 构建原则	87
7.3.5 部署原则	89
7.4 工作实践	90
7.5 小结	91
习题与思考题	92
扩展阅读与信息资源	92
第8章 理解需求	94
8.1 需求工程	95
8.2 建立根基	100
8.2.1 确认利益相关者	100
8.2.2 识别多重观点	100
8.2.3 协同合作	101
8.2.4 首次提问	101
8.2.5 非功能需求	102
8.2.6 可追溯性	102
8.3 获取需求	103
8.3.1 协作收集需求	103
8.3.2 质量功能部署	105
8.3.3 使用场景	106
8.3.4 获取工作产品	106
8.3.5 敏捷需求获取	107
8.3.6 面向服务的方法	107
8.4 开发用例	107
8.5 构建分析模型	111
8.5.1 分析模型的元素	112
8.5.2 分析模式	114
8.5.3 敏捷需求工程	114
8.5.4 自适应系统的需求	114
8.6 协商需求	115
8.7 需求监控	116
8.8 确认需求	117

8.9 避免常见错误	117	11.4 需求建模的模式	152
8.10 小结	118	11.4.1 发现分析模式	152
习题与思考题	118	11.4.2 需求模式举例：执行器－ 传感器	153
扩展阅读与信息资源	119	11.5 Web / 移动 App 的需求建模	156
第 9 章 需求建模：基于场景的 方法	121	11.5.1 多少分析才够用	156
9.1 需求分析	122	11.5.2 需求建模的输入	157
9.1.1 总体目标和原理	122	11.5.3 需求建模的输出	157
9.1.2 分析的经验原则	123	11.5.4 内容模型	158
9.1.3 域分析	123	11.5.5 WebApp 和移动 App 的 交互模型	159
9.1.4 需求建模的方法	125	11.5.6 功能模型	159
9.2 基于场景建模	126	11.5.7 WebApp 的配置模型	160
9.2.1 创建初始用例	126	11.5.8 导航建模	161
9.2.2 细化初始用例	128	11.6 小结	161
9.2.3 编写正式用例	129	习题与思考题	162
9.3 补充用例的 UML 模型	131	扩展阅读与信息资源	162
9.3.1 开发活动图	131	第 12 章 设计概念	163
9.3.2 泳道图	132	12.1 软件工程中的设计	164
9.4 小结	133	12.2 设计过程	166
习题与思考题	133	12.2.1 软件质量指导原则和 属性	166
扩展阅读与信息资源	133	12.2.2 软件设计的演化	168
第 10 章 需求建模：基于类的方法	135	12.3 设计概念	169
10.1 识别分析类	135	12.3.1 抽象	169
10.2 描述属性	138	12.3.2 体系结构	169
10.3 定义操作	138	12.3.3 模式	170
10.4 类－职责－协作者建模	140	12.3.4 关注点分离	170
10.5 关联和依赖	145	12.3.5 模块化	171
10.6 分析包	145	12.3.6 信息隐蔽	171
10.7 小结	146	12.3.7 功能独立	172
习题与思考题	146	12.3.8 求精	172
扩展阅读与信息资源	146	12.3.9 方面	173
第 11 章 需求建模：行为、模式和 Web / 移动 App	148	12.3.10 重构	173
11.1 生成行为模型	148	12.3.11 面向对象的设计概念	174
11.2 识别用例事件	149	12.3.12 设计类	174
11.3 状态表达	149	12.3.13 依赖倒置	176
		12.3.14 测试设计	177

12.4 设计模型	177	13.11 敏捷性与体系结构	203
12.4.1 数据设计元素	178	13.12 小结	204
12.4.2 体系结构设计元素	178	习题与思考题	205
12.4.3 接口设计元素	179	扩展阅读与信息资源	205
12.4.4 构件级设计元素	180		
12.4.5 部署级设计元素	181	第 14 章 构件级设计	207
12.5 小结	181	14.1 什么是构件	208
习题与思考题	182	14.1.1 面向对象的观点	208
扩展阅读与信息资源	183	14.1.2 传统的观点	209
		14.1.3 过程相关的观点	211
第 13 章 体系结构设计	184	14.2 设计基于类的构件	212
13.1 软件体系结构	185	14.2.1 基本设计原则	212
13.1.1 什么是体系结构	185	14.2.2 构件级设计指导方针	214
13.1.2 体系结构为什么重要	186	14.2.3 内聚性	215
13.1.3 体系结构描述	186	14.2.4 耦合性	216
13.1.4 体系结构决策	187	14.3 实施构件级设计	217
13.2 体系结构类型	188	14.4 WebApp 的构件级设计	222
13.3 体系结构风格	188	14.4.1 构件级内容设计	222
13.3.1 体系结构风格的简单 分类	189	14.4.2 构件级功能设计	222
13.3.2 体系结构模式	191	14.5 移动 App 的构件级设计	222
13.3.3 组织和求精	192	14.6 设计传统构件	223
13.4 体系结构考虑要素	192	14.7 基于构件的开发	223
13.5 体系结构决策	194	14.7.1 领域工程	223
13.6 体系结构设计	194	14.7.2 构件的合格性检验、适应 性修改与组合	224
13.6.1 系统环境的表示	195	14.7.3 体系结构不匹配	225
13.6.2 定义原型	195	14.7.4 复用的分析与设计	226
13.6.3 将体系结构细化为构件	196	14.7.5 构件的分类与检索	226
13.6.4 描述系统实例	197	14.8 小结	227
13.6.5 WebApp 的体系结构 设计	198	习题与思考题	228
13.6.6 移动 App 的体系结构 设计	198	扩展阅读与信息资源	228
13.7 评估候选的体系结构设计	199		
13.7.1 体系结构描述语言	200	第 15 章 用户界面设计	230
13.7.2 体系结构评审	201	15.1 黄金规则	231
13.8 经验学习	201	15.1.1 把控制权交给用户	231
13.9 基于模式的体系结构评审	202	15.1.2 减轻用户的记忆负担	232
13.10 体系结构一致性检查	203	15.1.3 保持界面一致	233
		15.2 用户界面的分析和设计	234
		15.2.1 用户界面分析和设计 模型	234

15.2.2 过程	235	16.6.2 设计粒度	266
15.3 界面分析	236	16.7 移动 App 模式	267
15.3.1 用户分析	236	16.8 小结	268
15.3.2 任务分析和建模	237	习题与思考题	268
15.3.3 显示内容分析	240	扩展阅读与信息资源	269
15.3.4 工作环境分析	240		
15.4 界面设计步骤	241	第 17 章 WebApp 设计	271
15.4.1 应用界面设计步骤	241	17.1 WebApp 设计质量	272
15.4.2 用户界面设计模式	243	17.2 设计目标	273
15.4.3 设计问题	243	17.3 WebApp 设计金字塔	274
15.5 WebApp 和移动 App 的界面设计	245	17.4 WebApp 界面设计	274
15.5.1 界面设计原则与指导方针	245	17.5 美学设计	275
15.5.2 WebApp 和移动 App 的界面设计工作流	248	17.5.1 布局问题	276
15.6 设计评估	249	17.5.2 平面设计问题	276
15.7 小结	250	17.6 内容设计	277
习题与思考题	250	17.6.1 内容对象	277
扩展阅读与信息资源	251	17.6.2 内容设计问题	278
		17.7 体系结构设计	278
第 16 章 基于模式的设计	253	17.7.1 内容体系结构	279
16.1 设计模式	254	17.7.2 WebApp 体系结构	280
16.1.1 模式的种类	255	17.8 导航设计	281
16.1.2 框架	256	17.8.1 导航语义	281
16.1.3 描述模式	257	17.8.2 导航语法	283
16.1.4 模式语言和模式库	258	17.9 构件级设计	283
16.2 基于模式的软件设计	258	17.10 小结	283
16.2.1 不同环境下基于模式的设计	258	习题与思考题	284
16.2.2 用模式思考	259	扩展阅读与信息资源	284
16.2.3 设计任务	260		
16.2.4 建立模式组织表	260	第 18 章 移动 App 设计	286
16.2.5 常见设计错误	261	18.1 挑战	287
16.3 体系结构模式	262	18.1.1 开发因素	287
16.4 构件级设计模式	263	18.1.2 技术因素	288
16.5 用户界面设计模式	264	18.2 开发移动 App	289
16.6 WebApp 设计模式	266	18.2.1 移动 App 质量	290
16.6.1 设计焦点	266	18.2.2 用户界面设计	291
		18.2.3 环境感知 App	292
		18.2.4 经验教训	293
		18.3 移动 App 设计的最佳实践	294

18.4 移动开发环境	295
18.5 云	297
18.6 传统软件工程的适用性	298
18.7 小结	298
习题与思考题	299
扩展阅读与信息资源	299

第三部分 质量管理

第 19 章 质量概念 302

19.1 什么是质量	303
19.2 软件质量	304
19.2.1 Garvin 的质量维度	304
19.2.2 McCall 的质量因素	305
19.2.3 ISO 9126 质量因素	306
19.2.4 定向质量因素	306
19.2.5 过渡到量化观点	307
19.3 软件质量困境	308
19.3.1 “足够好”的软件	308
19.3.2 质量的成本	309
19.3.3 风险	311
19.3.4 疏忽和责任	311
19.3.5 质量和安全	312
19.3.6 管理活动的影响	312
19.4 实现软件质量	313
19.4.1 软件工程方法	313
19.4.2 项目管理技术	313
19.4.3 质量控制	313
19.4.4 质量保证	313
19.5 小结	314
习题与思考题	314
扩展阅读与信息资源	314

第 20 章 评审技术 316

20.1 软件缺陷对成本的影响	317
20.2 缺陷的放大和消除	318
20.3 评审度量及其应用	319
20.3.1 分析度量数据	320
20.3.2 评审的成本效益	320

20.4 评审的正式程度	321
20.5 非正式评审	322
20.6 正式技术评审	323
20.6.1 评审会议	324
20.6.2 评审报告和记录保存	324
20.6.3 评审指导原则	325
20.6.4 样本驱动评审	326
20.7 产品完成后评估	327
20.8 小结	327
习题与思考题	327
扩展阅读与信息资源	328

第 21 章 软件质量保证 329

21.1 背景问题	330
21.2 软件质量保证的要素	330
21.3 软件质量保证的过程和产品特性	332
21.4 软件质量保证的任务、目标和度量	332
21.4.1 软件质量保证的任务	332
21.4.2 目标、属性和度量	333
21.5 软件质量保证的形式化方法	334
21.6 统计软件质量保证	335
21.6.1 一个普通的例子	335
21.6.2 软件工程中的六西格玛	336
21.7 软件可靠性	337
21.7.1 可靠性和可用性的测量	337
21.7.2 软件安全	338
21.8 ISO 9000 质量标准	339
21.9 软件质量保证计划	340
21.10 小结	341
习题与思考题	341
扩展阅读与信息资源	341

第 22 章 软件测试策略 343

22.1 软件测试的策略性方法	344
22.1.1 验证与确认	344
22.1.2 软件测试组织	345
22.1.3 软件测试策略——宏观	346

22.1.4 测试完成的标准	347	23.4.4 图矩阵	374
22.2 策略问题	348	23.5 控制结构测试	375
22.3 传统软件的测试策略	348	23.6 黑盒测试	376
22.3.1 单元测试	348	23.6.1 基于图的测试方法	376
22.3.2 集成测试	350	23.6.2 等价类划分	378
22.4 面向对象软件的测试策略	354	23.6.3 边界值分析	378
22.4.1 面向对象环境中的单元 测试	354	23.6.4 正交数组测试	379
22.4.2 面向对象环境中的集成 测试	354	23.7 基于模型的测试	381
22.5 WebApp 的测试策略	355	23.8 文档测试和帮助设施测试	381
22.6 移动 App 的测试策略	355	23.9 实时系统的测试	382
22.7 确认测试	356	23.10 软件测试模式	383
22.7.1 确认测试准则	356	23.11 小结	384
22.7.2 配置评审	356	习题与思考题	384
22.7.3 α 测试和 β 测试	356	扩展阅读与信息资源	385
22.8 系统测试	358	第 24 章 测试面向对象的应用	386
22.8.1 恢复测试	358	24.1 扩展测试的视野	387
22.8.2 安全测试	358	24.2 测试 OOA 和 OOD 模型	387
22.8.3 压力测试	358	24.2.1 OOA 和 OOD 模型的 正确性	388
22.8.4 性能测试	359	24.2.2 面向对象模型的一致性	388
22.8.5 部署测试	359	24.3 面向对象测试策略	389
22.9 调试技巧	360	24.3.1 面向对象环境中的单元 测试	389
22.9.1 调试过程	360	24.3.2 面向对象环境中的集成 测试	390
22.9.2 心理因素	361	24.3.3 面向对象环境中的确认 测试	390
22.9.3 调试策略	362	24.4 面向对象测试方法	390
22.9.4 纠正错误	363	24.4.1 面向对象概念的测试用例 设计含义	391
22.10 小结	363	24.4.2 传统测试用例设计方法的 可应用性	391
习题与思考题	364	24.4.3 基于故障的测试	391
扩展阅读与信息资源	364	24.4.4 基于场景的测试设计	392
第 23 章 测试传统的应用软件	366	24.5 类级可应用的测试方法	392
23.1 软件测试基础	367	24.5.1 面向对象类的随机测试	392
23.2 测试的内部视角和外部视角	368	24.5.2 类级的划分测试	393
23.3 白盒测试	369	24.6 类间测试用例设计	394
23.4 基本路径测试	369		
23.4.1 流图表示	369		
23.4.2 独立程序路径	371		
23.4.3 生成测试用例	372		

24.6.1 多类测试	394	26.2 测试策略	420
24.6.2 从行为模型导出的测试	395	26.2.1 传统方法适用吗	420
24.7 小结	396	26.2.2 对自动化的要求	421
习题与思考题	396	26.2.3 建立测试矩阵	422
扩展阅读与信息资源	397	26.2.4 压力测试	422
26.2.5 生产环境中的测试	423	26.3 与用户交互的各种情况	424
第 25 章 测试 WebApp	398	26.3.1 手语测试	425
25.1 WebApp 的测试概念	398	26.3.2 语音输入和识别	425
25.1.1 质量维度	399	26.3.3 虚拟键盘输入	425
25.1.2 WebApp 环境中的错误	399	26.3.4 警报和异常条件	426
25.1.3 测试策略	400	26.4 跨界测试	426
25.1.4 测试计划	400	26.5 实时测试问题	427
25.2 测试过程概述	401	26.6 测试工具和环境	427
25.3 内容测试	402	26.7 小结	428
25.3.1 内容测试的目标	402	习题与思考题	429
25.3.2 数据库测试	403	扩展阅读与信息资源	430
25.4 用户界面测试	404	第 27 章 安全性工程	431
25.4.1 界面测试策略	404	27.1 安全性需求分析	432
25.4.2 测试界面机制	405	27.2 网络世界中的安全性与 保密性	433
25.4.3 测试界面语义	406	27.2.1 社交媒体	433
25.4.4 可用性测试	406	27.2.2 移动 App	434
25.4.5 兼容性测试	408	27.2.3 云计算	434
25.5 构件级测试	409	27.2.4 物联网	434
25.6 导航测试	409	27.3 安全性工程分析	434
25.6.1 测试导航语法	409	27.3.1 安全性需求获取	435
25.6.2 测试导航语义	410	27.3.2 安全性建模	435
25.7 配置测试	411	27.3.3 测度设计	436
25.7.1 服务器端问题	411	27.3.4 正确性检查	436
25.7.2 客户端问题	411	27.4 安全性保证	437
25.8 安全性测试	412	27.4.1 安全性保证过程	437
25.9 性能测试	413	27.4.2 组织和管理	438
25.9.1 性能测试的目标	413	27.5 安全性风险分析	438
25.9.2 负载测试	414	27.6 传统软件工程活动的作用	440
25.9.3 压力测试	414	27.7 可信性系统验证	441
25.10 小结	415	27.8 小结	442
习题与思考题	416	习题与思考题	443
扩展阅读与信息资源	417	扩展阅读与信息资源	443
第 26 章 测试移动 App	418		
26.1 测试准则	419		

第 28 章 形式化建模与验证	445	29.4.2 配置对象	474
28.1 净室策略	446	29.4.3 内容管理	475
28.2 功能规格说明	447	29.4.4 变更管理	477
28.2.1 黑盒规格说明	448	29.4.5 版本控制	479
28.2.2 状态盒规格说明	449	29.4.6 审核和报告	479
28.2.3 清晰盒规格说明	449	29.5 小结	480
28.3 净室设计	449	习题与思考题	481
28.3.1 设计细化	450	扩展阅读与信息资源	481
28.3.2 设计验证	450	第 30 章 产品度量	483
28.4 净室测试	451	30.1 产品度量框架	484
28.4.1 统计使用测试	451	30.1.1 测度、度量和指标	484
28.4.2 认证	452	30.1.2 产品度量的挑战	484
28.5 重新思考形式化方法	453	30.1.3 测量原则	485
28.6 形式化方法的概念	454	30.1.4 面向目标的软件测量	485
28.7 其他争论	457	30.1.5 有效软件度量的属性	486
28.8 小结	457	30.2 需求模型的度量	487
习题与思考题	458	30.2.1 基于功能的度量	487
扩展阅读与信息资源	459	30.2.2 规格说明质量的度量	490
第 29 章 软件配置管理	460	30.3 设计模型的度量	491
29.1 软件配置管理概述	461	30.3.1 体系结构设计的度量	491
29.1.1 SCM 场景	461	30.3.2 面向对象设计的度量	493
29.1.2 配置管理系统的元素	462	30.3.3 面向类的度量——CK 度量集	493
29.1.3 基线	463	30.3.4 面向类的度量——MOOD 度量集	495
29.1.4 软件配置项	464	30.3.5 Lorenz 和 Kidd 提出的面向 对象的度量	496
29.1.5 依赖性和变更管理	464	30.3.6 构件级设计的度量	496
29.2 SCM 中心存储库	465	30.3.7 面向操作的度量	496
29.2.1 一般特征和内容	465	30.3.8 用户界面设计的度量	497
29.2.2 SCM 特征	466	30.4 WebApp 和移动 App 的设计 度量	497
29.3 SCM 过程	466	30.5 源代码的度量	499
29.3.1 软件配置中的对象标识	467	30.6 测试的度量	500
29.3.2 版本控制	468	30.6.1 用于测试的 Halstead 度量	500
29.3.3 变更控制	469	30.6.2 面向对象测试的度量	500
29.3.4 影响管理	471	30.7 维护的度量	501
29.3.5 配置审核	472		
29.3.6 状态报告	472		
29.4 WebApp 和移动 App 的配置 管理	473		
29.4.1 主要问题	473		

30.8 小结	502
习题与思考题	503
扩展阅读与信息资源	503

第四部分 管理软件项目

第 31 章 项目管理概念	506
31.1 管理涉及的范围	507
31.1.1 人员	507
31.1.2 产品	507
31.1.3 过程	508
31.1.4 项目	508
31.2 人员	508
31.2.1 利益相关者	508
31.2.2 团队负责人	509
31.2.3 软件团队	509
31.2.4 敏捷团队	511
31.2.5 协调和沟通问题	512
31.3 产品	513
31.3.1 软件范围	513
31.3.2 问题分解	513
31.4 过程	514
31.4.1 合并产品和过程	514
31.4.2 过程分解	515
31.5 项目	516
31.6 W ⁵ HH 原则	516
31.7 关键实践	517
31.8 小结	518
习题与思考题	518
扩展阅读与信息资源	519

第 32 章 过程度量与项目度量

32.1 过程领域和项目领域中的 度量	522
32.1.1 过程度量和软件过程 改进	522
32.1.2 项目度量	524
32.2 软件测量	525
32.2.1 面向规模的度量	526

32.2.2 面向功能的度量	527
32.2.3 调和代码行度量和功能点 度量	527
32.2.4 面向对象的度量	528
32.2.5 面向用例的度量	529
32.2.6 WebApp 项目的度量	530
32.3 软件质量的度量	531
32.3.1 测量质量	531
32.3.2 缺陷排除效率	532
32.4 在软件过程中集成度量	533
32.4.1 支持软件度量的论点	534
32.4.2 建立基线	534
32.4.3 度量收集、计算和评估	534
32.5 小型组织的度量	535
32.6 制定软件度量大纲	536
32.7 小结	537
习题与思考题	537
扩展阅读与信息资源	538

第 33 章 软件项目估算

33.1 对估算的观察	541
33.2 项目计划过程	542
33.3 软件范围和可行性	542
33.4 资源	543
33.4.1 人力资源	544
33.4.2 可复用软件资源	544
33.4.3 环境资源	544
33.5 软件项目估算	544
33.6 分解技术	545
33.6.1 软件规模估算	545
33.6.2 基于问题的估算	546
33.6.3 基于 LOC 估算的实例	547
33.6.4 基于 FP 估算的实例	548
33.6.5 基于过程的估算	549
33.6.6 基于过程估算的实例	550
33.6.7 基于用例的估算	550
33.6.8 基于用例点估算的实例	551
33.6.9 调和不同的估算方法	552
33.7 经验估算模型	553

33.7.1 估算模型的结构	553	35.2 软件风险	580
33.7.2 COCOMO II 模型	553	35.3 风险识别	581
33.7.3 软件方程	553	35.3.1 评估整体项目风险	582
33.8 面向对象项目的估算	554	35.3.2 风险因素和驱动因子	583
33.9 特殊的估算技术	555	35.4 风险预测	584
33.9.1 敏捷开发的估算	555	35.4.1 建立风险表	584
33.9.2 WebApp 项目的估算	555	35.4.2 评估风险影响	585
33.10 自行开发或购买的决策	556	35.5 风险细化	587
33.10.1 创建决策树	557	35.6 风险缓解、监测和管理	587
33.10.2 外包	558	35.7 RMMM 计划	589
33.11 小结	559	35.8 小结	590
习题与思考题	559	习题与思考题	590
扩展阅读与信息资源	560	扩展阅读与信息资源	591
第 34 章 项目进度安排	561	第 36 章 维护与再工程	593
34.1 基本概念	562	36.1 软件维护	594
34.2 项目进度安排概述	563	36.2 软件可支持性	595
34.2.1 基本原则	564	36.3 再工程	596
34.2.2 人员与工作量之间的 关系	565	36.4 业务过程再工程	596
34.2.3 工作量分配	566	36.4.1 业务过程	596
34.3 为软件项目定义任务集	567	36.4.2 BPR 模型	597
34.3.1 任务集举例	567	36.5 软件再工程	598
34.3.2 主要任务的细化	568	36.5.1 软件再工程过程模型	598
34.4 定义任务网络	568	36.5.2 软件再工程活动	599
34.5 进度安排	569	36.6 逆向工程	600
34.5.1 时序图	570	36.6.1 理解数据的逆向工程	601
34.5.2 跟踪进度	570	36.6.2 理解处理的逆向工程	602
34.5.3 跟踪面向对象项目的 进展	572	36.6.3 用户界面的逆向工程	602
34.5.4 WebApp 和移动 App 项目的 进度安排	573	36.7 重构	603
34.6 挣值分析	575	36.7.1 代码重构	603
34.7 小结	576	36.7.2 数据重构	604
习题与思考题	577	36.8 正向工程	604
扩展阅读与信息资源	578	36.8.1 客户 / 服务器体系结构的 正向工程	605
第 35 章 风险管理	579	36.8.2 面向对象体系结构的正向 工程	606
35.1 被动风险策略和主动风险 策略	580	36.9 再工程经济学	606
		36.10 小结	607
		习题与思考题	607

扩展阅读与信息资源	608
-----------------	-----

第五部分 软件工程高级课题

第 37 章 软件过程改进	612
---------------------	-----

37.1 什么是 SPI	613
--------------------	-----

37.1.1 SPI 的方法	613
----------------------	-----

37.1.2 成熟度模型	614
--------------------	-----

37.1.3 SPI 适合每个人吗	615
-------------------------	-----

37.2 SPI 过程	615
-------------------	-----

37.2.1 评估和差距分析	616
----------------------	-----

37.2.2 教育和培训	617
--------------------	-----

37.2.3 选择和合理性判定	617
-----------------------	-----

37.2.4 设置 / 迁移	618
----------------------	-----

37.2.5 评价	618
-----------------	-----

37.2.6 SPI 的风险管理	618
------------------------	-----

37.3 CMMI	619
-----------------	-----

37.4 人员 CMM	622
-------------------	-----

37.5 其他 SPI 框架	622
----------------------	-----

37.6 SPI 的投资收益率	624
-----------------------	-----

37.7 SPI 趋势	624
-------------------	-----

37.8 小结	625
---------------	-----

习题与思考题	626
--------------	-----

扩展阅读与信息资源	626
-----------------	-----

第 38 章 软件工程的新趋势	627
-----------------------	-----

38.1 技术演变	628
-----------------	-----

38.2 关于纯粹工程学科的展望	629
------------------------	-----

38.3 观察软件工程的的发展趋势	629
-------------------------	-----

38.4 识别“软趋势”	630
--------------------	-----

38.4.1 管理复杂性	631
--------------------	-----

38.4.2 开放世界软件	632
---------------------	-----

38.4.3 意外需求	632
-------------------	-----

38.4.4 人才技能结合	633
---------------------	-----

38.4.5 软件构造块	633
--------------------	-----

38.4.6 对“价值”认识的转变	634
-------------------------	-----

38.4.7 开源	634
-----------------	-----

38.5 技术方向	634
-----------------	-----

38.5.1 过程趋势	634
-------------------	-----

38.5.2 巨大的挑战	635
--------------------	-----

38.5.3 协同开发	636
-------------------	-----

38.5.4 需求工程	636
-------------------	-----

38.5.5 模型驱动的软件开发	637
------------------------	-----

38.5.6 后现代设计	637
--------------------	-----

38.5.7 测试驱动的开发	638
----------------------	-----

38.6 相关工具的趋势	639
--------------------	-----

38.7 小结	640
---------------	-----

习题与思考题	640
--------------	-----

扩展阅读与信息资源	640
-----------------	-----

第 39 章 结束语	642
------------------	-----

39.1 再论软件的重要性	643
---------------------	-----

39.2 人员及其构造系统的方式	643
------------------------	-----

39.3 表示信息的新模式	644
---------------------	-----

39.4 远景	645
---------------	-----

39.5 软件工程师的责任	646
---------------------	-----

39.6 写在最后	647
-----------------	-----

索引	648
----------	-----

在线资源^①

附录 1 UML 简介

附录 2 面向对象概念

附录 3 形式化方法

参考文献

① 请访问华章网站 (www.hzbook.com) 下载在线资源。

软件的本质

要点浏览

概念: 计算机软件是由专业人员开发并长期维护的软件产品。完整的软件产品包括: 可以在各种不同容量及系统结构的计算机上运行的程序、程序运行过程中产生的各种结果以及各种描述信息, 这些信息可以以硬拷贝或是各种电子媒介形式存在。

人员: 软件工程师开发软件并提供技术支持, 产业界中几乎每个人都间接或直接地使用软件。

重要性: 软件之所以重要是因为它在我们的生活中无所不在, 并且日渐深入到商业、文化和日常生活的各个方面。

步骤: 客户和利益相关者表达对计算机软件的要求, 工程师构建软件产品, 最终用户应用软件来解决特定的问题或者满足特定的要求。

工作产品: 在一种或多种特定环境中运行并服务于一个或多个最终用户要求的计算机软件。

质量保证措施: 如果你是软件工程师, 就要应用本书中包含的思想。如果你是最终用户, 应确保理解了自己的要求和环境, 然后选择能很好地满足两者的应用软件。

在给我演示了最新开发的世界上最流行的第一人称射击视频游戏之后, 这位年轻的开发者笑了。

“你不是一个玩家, 对吗?” 他问道。

我微笑道: “你是怎么猜到的?”

这位年轻人身着短裤和T恤衫。他的腿像活塞那样上下跳动, 燃烧着神经能量, 这在他的同事中看起来是很平常的。

“因为如果你是玩家,” 他说, “你应该会更加兴奋。你已经看到了我们的下一代产品, 我们的客户会对它着迷……这不是开玩笑。”

我们坐在开发区, 这是地球上最成功的游戏开发者之一。在过去几年中, 他演示的前几代游戏售出了 5000 万份, 收入达几亿美元。

“那么, 这一版什么时候上市?” 我问道。

他耸耸肩, “大约在 5 个月以内, 我们还有很多工作要做。”

他负责一个应用软件中的游戏和人工智能功能, 该软件包含的代码超过了 300 万行。

“你们使用任何软件工程技术吗?” 我问道, 估计他会笑笑并摇头。

他停顿了一下, 想了一会儿, 然后缓慢地点点头。“我们让软件工程技术适应我们的需求, 但是, 我们确实使用。”

“在什么地方使用?” 我试探地问道。

关键概念

应用领域

云计算

失效曲线

遗留软件

移动 App

产品线

软件定义

软件问题

软件的本质

磨损

WebApp

“我们的问题是经常将需求翻译成创意。”

“创意？”我打断了他的话。

“你知道，那些设计故事、人物及所有游戏素材的家伙，他们想的是游戏大卖。而我们不得不接受他们抛给我们的这些，并形成一组技术需求，从而构建游戏。”

“那么形成了需求之后呢？”

他耸耸肩，“我们不得不扩展并修改以前游戏版本的体系结构，并创建新的产品。我们需要根据需求创建代码，对每日构建的代码实施测试，并且做你书中建议的很多事情。”

“你知道我的书？”老实说，我非常惊讶。

“当然，在学校使用。那里有很多。”

“我已经与你的很多同事谈了，他们对我书中的很多东西持怀疑态度。”

他皱了皱眉，“你看，我们不是 IT 部门或航空公司，所以我们不得不对你所提倡的东西进行取舍。但是底线是一样的——我们需要生产高质量的产品，并且以可重复方式完成这一目标的唯一途径是改写我们自己的软件工程技术子集。”

“那么这个子集是如何随着时间的推移变更的？”

他停顿了一下，像是在思考着未来。“游戏将变得更加庞大和复杂，那是肯定的。随着更多竞争的出现，我们的开发时间表将会收缩。慢慢地，游戏本身会迫使我们应用更多的开发规范。如果我们不这样做，我们就会死掉。”

计算机软件仍然是世界舞台上最为重要的技术，并且也是“意外效应法则”的典型例子。60年前，没有人曾预料到软件科学会成为今天商业、科学和工程所必需的技术。软件促进了新科技的创新（例如基因工程和纳米科技）、现代科技的发展（例如通信）以及传统技术的根本转变（例如媒体行业），软件技术已经成为个人计算机革命的推动力量，消费者使用智能手机就可以购买软件产品。软件还将由产品逐渐演化为服务，软件公司随需应变，通过 Web 浏览器发布即时更新功能，软件公司几乎可以比所有工业时代的公司都更大、更有影响力。在大量应用软件的驱动下，互联网将迅速发展，并逐渐改变人们生活的诸多方面——从图书馆搜索、消费购物、政治论战到年轻人和（不很年轻的）成年人的约会行为。

引述 创新观念和科技发现是经济增长的推进器。
——《华尔街日报》

没有人曾想到软件可嵌入各种系统中：交通运输、医疗、通信、军事、工业、娱乐以及办公设备……不胜枚举。如果笃信“意外效应法则”的话，那么还有很多结果和影响是我们尚未预料到的。

没有人曾想到，随着时间的推移，将有数百万的计算机程序需要进行纠错、适应性调整和优化，这些维护工作将耗费比开发新软件更多的人力和物力。

随着软件重要性的日渐凸现，软件业界一直试图开发新的技术，使得高质量计算机程序的开发和维护更容易、更快捷、成本更低廉。一些技术主要针对特定应用领域（例如网站设计和实现），另一些着眼于技术领域（例如面向对象系统、面向方面的程序设计），还有一些覆盖面很宽（例如像 LINUX 这样的操作系统）。然而，我们尚未开发出一种可以实现上述所有需求的软件技术，而且未来能够产生这种技术的可能性也很小。人们也尚未将其工作、享受、安全、娱乐、决策以及全部生活都完全依赖于计算机软件。这或许是正确的选择。

本书为需要构建正确软件的计算机软件工程师提供了一个框架。该框架包括过程、一系列方法以及我们称为软件工程的工具。

1.1 软件的本质

现在的软件具有产品和产品交付载体的双重作用。作为产品，软件显示了由计算机硬件体现的计算能力，更广泛地说，显示的是由一个可被本地硬件设备访问的计算机网络体现的计算潜力。无论是安装在移动电话、手持平板电脑、台式机还是大型计算机中，软件都扮演着信息转换的角色：产生、管理、获取、修改、显示或者传输各种不同的信息，简单如几个比特的传递，复杂如从多个独立的数据源获取的多媒体演示。而作为产品生产的载体，软件提供了计算机控制（操作系统）、信息通信（网络）以及应用程序开发和控制（软件工具和环境）的基础平台。

软件提供了我们这个时代最重要的产品——信息。它转换个人数据（例如个人财务交易），从而使信息在一定范围内发挥更大的作用；它通过管理商业信息提升竞争力；它为世界范围的信息网络提供通路（例如因特网），并为各类格式的信息提供不同的查询方式。软件还提供了可以威胁个人隐私的载体，并给那些怀有恶意目的的人提供了犯罪的途径。

在最近半个世纪里，计算机软件的作用发生了很大的变化。硬件性能的极大提高、计算机结构的巨大变化、存储容量的大幅度增加以及种类繁多的输入和输出方法都促使基于计算机的系统更加先进和复杂。如果系统开发成功，那么“先进和复杂”可以产生惊人的效果，但是同时复杂性也给系统的开发人员和防护人员带来巨大的挑战。

现在，一个庞大的软件产业已经成为了工业经济中的主导因素。早期的独立程序员也已经被专业的软件开发团队所代替，团队中的不同专业技术人员可分别关注复杂应用系统中的某一部分技术。然而同过去的独立程序员一样，开发现代计算机系统时，软件开发人员依然面临同样的问题：^①

- 为什么软件需要如此长的开发时间？
- 为什么开发成本居高不下？
- 为什么在将软件交付顾客使用之前，我们无法找到所有的错误？
- 为什么维护已有的程序要花费如此多的时间和人工？
- 为什么软件开发和维护的进度仍旧难以度量？

种种问题显示了业界对软件以及软件开发方式的关注，这种关注导致了业界对软件工程实践方法的采纳。

1.1.1 定义软件

今天，绝大多数专业人员和许多普通人认为他们对软件大概了解。真的是这样吗？

来自教科书的关于软件的定义也许是：

软件是：（1）指令的集合（计算机程序），通过执行这些指令可以满足预期的特性、功能和性能需求；（2）数据结构，使得程序可以合理利用信息；（3）软件描述信息，它以硬拷贝和虚拟形式存在，用来描述程序的操作和使用。

关键点 软件既是产品也是交付产品的载体。

引述 软件是播撒梦想和收获噩梦的地方，是一片恶魔和神仙竞争的抽象而神秘的沼泽，是一个狼人和银弹共存的矛盾世界。

Brad J.Cox

3

^① 在一本优秀的关于软件业务的论文集中，Tom DeMarco[DeM95]提出了相反的看法。他认为：“我们更应该总结使得当今的软件开发费用低廉的成功经验，而不是不停地质问为何软件开发成本高昂。这会有助于我们继续保持软件产业的杰出成就。”

当然，还有更完整的解释。但是一个更加正式的定义可能并不能显著改善其可理解性。为了更好地理解“软件”的含义，有必要将软件和其他人工产品的特点加以区分。软件是逻辑的而非物理的系统元素。因此，软件和硬件具有完全不同的特性：软件不会“磨损”。

图 1-1 描述了硬件的失效率，该失效率是时间的函数。这个名为“浴缸曲线”的关系图显示：硬件在早期具有相对较高的失效率（这种失效通常来自设计或生产缺陷）；在缺陷被逐个纠正之后，失效率随之降低并在一段时间内保持平稳（理想情况下很低）；然而，随着时间推移，因为灰尘、振动、不当使用、温度超限以及其他环境问题所造成的硬件组件损耗累积的效果，使得失效率再次抬高。简而言之，硬件开始磨损了。

建议 若希望降低软件退化，则需要改进软件的设计（第 12～18 章）。

而软件是不会被引起硬件磨损的环境问题所影响的。因此，从理论上来说，软件的失效率曲线应该呈现为图 1-2 的“理想曲线”。未知的缺陷将在程序生命周期的前期造成高失效率。然而随着错误的纠正，曲线将如图中所示趋于平缓。“理想曲线”只是软件实际失效模型的粗略简化。曲线的含义很明显——软件不会磨损，但是软件退化的确存在。

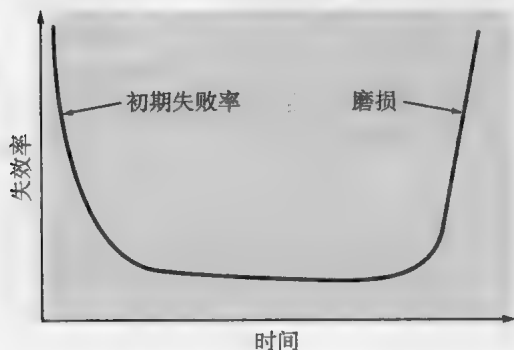


图 1-1 硬件失效曲线图

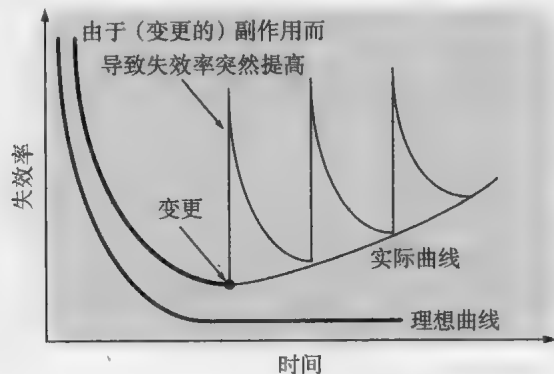


图 1-2 软件失效曲线图

这个似乎矛盾的现象用图 1-2 所示的“实际曲线”可以很好地解释。在完整的生命周期里，^①软件将会面临变更，每次变更都可能引入新的错误，使得失效率像“实际曲线”（图 1-2）那样陡然上升。在曲线回到最初的稳定失效率状态前，新的变更会引起曲线又一次上升。就这样，最小的失效率点沿类似于斜线的形状逐渐上升，可以说，不断的变更是软件退化的根本原因。

关键点 软件工程方法的目的是降低图 1-2 中曲线向上突变的幅度及实际失效曲线的斜率。

磨损的另一方面同样说明了软硬件的不同。磨损的硬件部件可以用备用部件替换，而软件却不存在备用部件。每个软件的缺陷都暗示了设计的缺陷或者在从设计转化到机器可执行代码的过程中产生的错误。因此，软件维护要应对变更请求，比硬件维护更为复杂。

1.1.2 软件应用领域

今天，计算机软件可分为七个大类，软件工程师正面临持续的挑战。

系统软件——一整套服务于其他程序的程序。某些系统软件（例如编译器、编辑器、文

^① 事实上，在软件开发开始，在第一个版本发布之前的很长时间，许多利益相关者可能提出变更要求。

件管理软件)处理复杂但确定的^①信息结构,另一些系统应用程序(例如操作系统构件、驱动程序、网络软件、远程通信处理器)主要处理的是不确定的数据。

应用软件——解决特定业务需要的独立应用程序。这类应用软件处理商务或技术数据,以协助业务操作或协助做出管理或技术决策。

工程/科学软件——“数值计算”(number crunching)类程序涵盖了广泛的应用领域,从天文学到火山学,从自动压力分析到轨道动力学,从计算机辅助设计到分子生物学,从遗传分析到气象学。

嵌入式软件——嵌入式软件存在于某个产品或者系统中,可实现和控制面向最终用户和系统本身的特性和功能。嵌入式软件可以执行有限的和内部的功能(例如微波炉的按键控制),或者提供重要的功能和控制能力(例如汽车中的燃油控制、仪表板显示、刹车系统等汽车电子功能)。

产品线软件——为多个不同用户的使用提供特定功能。产品线软件关注有限的及内部的市场(例如库存控制产品)或者大众消费品市场。

Web/移动App——以网络为中心,其概念涵盖了宽泛的应用软件产品,包括基于浏览器的App和安装在移动设备上的软件。

人工智能软件——利用非数值算法解决计算和直接分析无法解决的复杂问题。这个领域的应用程序包括机器人、专家系统、模式识别(图像和语音)、人工神经网络、定理证明和博弈等。

全世界成百万的软件工程师在为以上各类软件项目努力地工作着。有时是建立一个新的系统,而有时只是对现有应用程序的纠错、适应性调整和升级。一个年轻的软件工程师所经手项目本身的年限比他自己的年龄还大是常有的事。对于上述讨论的各类软件,上一代的软件工程师都已经留下了遗留系统。我们希望这代工程师留下的遗留系统可以减轻未来工程师的负担。

网络资源 目前最大的共享软件/免费软件库之一: shareware.cnet.com。

引述 对于我来说,电脑是我们能够想出的最重要的工具。它相当于思想的自行车。

Steve Jobs

1.1.3 遗留软件

成千上万的计算机程序都可以归于在前一小节中讨论的7大类应用领域。其中某些是当今最先进的软件——最近才对个人、企业和政府发布。但是另外一些软件则年代较久,甚至过于久远了。

这些旧的系统——通常称为遗留软件(legacy software)——从20世纪60年代起,就成为人们持续关注的焦点。Dayani-Fard和他的同事[Day99]这样描述遗留软件:

遗留软件系统……在几十年前诞生,它们不断被修改以满足商业需要和计算平台的变化。这类系统的繁衍使得大型机构十分头痛,因为它们为维护代价高昂且系统演化风险较高。

Liu和他的同事[Liu98]进一步扩展了这个描述,指出“许多遗留软件系统仍然支持核心的商业功能,是业务必不可少的支撑。”因此,遗留软件具有生命周期长以及业务关键性的特点。

然而不幸的是,遗留软件常常存在另一个特点——质量差^②。遗留软件通常拥有数不清的

① 软件的确定性是指系统的输入、处理和输出的顺序及时间是可以预测的,软件的不确定性是指系统的输入、处理和输出的顺序和时间是无法提前预测的。

② 所谓“质量差”是基于现代软件工程思想的,这个评判标准对遗留系统有些不公平,因为在遗留软件开发的年代里,现代的软件工程一些概念和原则可能还没有被人们完全理解。

问题：遗留系统的设计难以扩展，代码令人费解，文档混乱甚至根本没有，测试用例和结果并未归档，变更的历史管理混乱……然而，这些系统仍然支撑着“核心的业务功能，并且是业务必不可少的支撑”。该如何应对这种情况？

提问 如果遇到质量低下的遗留软件该怎么办？

最合理的回答也许就是什么也不做，至少在其不得不进行重大变更之前什么也不做。如果遗留软件可以满足用户的需求并且能可靠运行，那么它就没有失效，不需要修改。然而，随着时间的推移，遗留系统经常会由于下述原因而发生演化：

提问 对遗留软件都进行了哪些类型的改变？

- 软件需要进行适应性调整，从而可以满足新的计算环境或者技术的需求。
- 软件必须升级以实现新的商业需求。
- 软件必须扩展以使之具有与更多新的系统和数据库的互操作能力。
- 软件架构必须进行改建以使之能适应不断演化的计算环境。

建议 所有软件工程师都需认识到变更是不可避免的。不要反对变更。

当这些变更发生时，遗留系统需要经过再工程（第36章）以适应未来的多样性。当代软件工程的目标是“修改在进化论理论上建立的方法论”，即“软件系统不断经历变更，新的软件系统从旧系统中建立起来，并且……新旧所有系统都必须具有互操作性和协作性。” [Day99]。

1.2 软件的变更本质

四大类软件不断演化，在行业中占有主导地位。这四类软件在十几年前还处于初级阶段。

1.2.1 WebApp

万维网（WWW）的早期（大约从1990年到1995年），Web站点仅包含链接在一起的一些超文本文件，这些文件使用文本和有限的图形来表示信息。随着时间的推移，一些开发工具（例如XML、Java）扩展了HTML（超级文本标记语言）的能力，使得Web工程师在向客户提供信息的同时也能提供计算能力。基于Web的系统和应用软件^①（我们将这些总称为WebApp）诞生了。

引述 对于任何类型的稳定性，Web都会变成完全不同的东西。

Louis Monier

今天，WebApp已经发展成为成熟的计算工具，这些工具不仅可以为最终用户提供独立的功能，而且已经同公司数据库和业务应用系统集成在一起了。

十年前，WebApp“演化成为一种混合体，介于印刷出版和软件开发之间、市场和计算之间内部通信和外部关系之间以及艺术和技术之间。” [Pow98]，但是，如1.1.2节所述，当前WebApp在很多应用类型中提供了丰富的计算能力。

在过去的十多年中，语义Web技术（通常指Web 3.0）已经演化为成熟的企业和消费者应用软件，包括“提供新功能的语义数据库，这些新功能需要Web链接、灵活的数据表示以及外部访问API（应用编程接口）。” [Hen10] 成熟的关系型数据结构导致了全新的WebApp，允许以多种方式访问不同的信息，这在以前是不可能做到的。

① 在本书中，WebApp这个术语包含了很多事物，从一个简单的帮助消费者计算汽车租赁费用的网页，到为商务旅行和度假提供全套旅游服务的大型复杂的Web站点。其中包括完整的Web站点、Web站点的专门功能以及在Internet、Intranet或Extranet上的信息处理应用软件。

1.2.2 移动 App

术语 App 已经演化为在移动平台（例如 iOS、Android 或 Windows Mobile）上专门设计的软件。在大多数情况下，移动 App 包括用户接口，用户接口利用移动平台所提供的独特的交互机制，基于 Web 资源的互操作性提供与 App 相关的大量信息的访问，并具有本地处理能力，以最适合移动平台的方式收集、分析和格式化信息。此外，移动 App 提供了在平台中的持久存储能力。

认识到移动 WebApp 与移动 App 之间的微妙差异是非常重要的。移动 WebApp 允许移动设备通过针对移动平台的优点和弱点专门设计的浏览器获取基于 Web 内容的访问。移动 App 可以直接访问设备的硬件特性（例如加速器或者 GPS 定位），然后提供前面所述的本地处理和存储能力。随着时间的推移，移动浏览器会变得更加成熟，并可获取对设备级硬件和信息的访问，这将使得移动 WebApp 和移动 App 之间的区别变得模糊。

提问 WebApp 和移动 App 之间的差别是什么？

1.2.3 云计算

云计算包括基础设施或“生态系统”，它能使得任何用户在任何地点都可以使用计算设备来共享广泛的计算资源。云计算的总体逻辑结构如图 1-3 所示。



图 1-3 云计算的逻辑结构 [Wik13]

如图所示，计算设备位于云的外部，可以访问云内的各种资源。这些资源包括应用软件、平台和基础设施。最简单的形式是，外部计算设备通过 Web 浏览器或类似的软件访问云。云提供对存储在数据库或其他数据结构中的数据的访问。此外，设备可访问可执行的应

用软件，可以用这种应用程序代替计算设备上的 App。

云计算的实现需要开发包含前端和后端服务的体系结构。前端包括客户（用户）设备和应用软件（如浏览器），用于访问后端。后端包括服务器和相关的计算资源、数据存储系统（如数据库）、服务器驻留应用程序和管理服务器。通过建立对云及其驻留资源的一系列访问协议，管理服务器使用中间件对流量进行协调和监控。[Str08]

可以对云体系结构进行分段，以提供不同级别的访问，从公共访问到只对授权用户提供访问的私有云体系结构。

1.2.4 产品线软件

美国卡内基·梅隆大学软件工程研究所（SEI）将软件产品线定义为“一系列软件密集型系统，可以共享一组公共的可管理的特性，这些特性可以满足特定市场或任务的特定需求，并以预定的方法从一组公共的核心资源开发出来。”[SEI13]以某种方式使软件产品相互关联，可见，软件产品线的概念并不是新概念。但是，软件产品线的思想提供了重要的工程影响力，软件产品线都使用相同的底层应用软件和数据体系结构来开发，并使用可在整个产品线中进行复用的一组软件构件来实现。

软件产品线共享一组资源，包括需求（第8章）、体系结构（第13章）、设计模式（第16章）、可重用构件（第14章）、测试用例（第22、23章）及其他的软件工作产品。本质上，软件产品线是对很多产品进行开发的结果，在对这些产品进行工程设计时，利用了产品线中所有产品的公共性。

1.3 小结

软件是以计算机为基础的系统和产品中的关键部分，并且是世界舞台上最重要的技术之一。在过去的50年里，软件已经从解决特定问题和信息分析的工具发展为独立的产业。然而，如何在有限的时间内利用有限的资金开发高质量的软件，这仍然是我们所面对的难题。

11

软件——程序、数据和描述信息——覆盖了科技和应用的很多领域。遗留软件仍旧给维护人员带来了特殊的挑战。

软件的本质是变更。基于 Web 的系统和 App 已经从简单的信息内容集合演化为能够展示复杂功能和多媒体信息的复杂系统。尽管 WebApp 具有独特的特性和需求，但它们仍然属于软件范畴。由于 App 已迁移到很多的平台上，因此移动 App 展示出了新的挑战。云计算将转变软件交付的方式及软件存在的环境。产品线软件提供了构建软件的潜在效率。

习题与思考题

- 1.1. 举出至少5个例子来说明“意外效应法则”在计算机软件方面的应用。
- 1.2. 举例说明软件对社会的影响（包括正面影响和负面影响）。
- 1.3. 针对1.1节提出的5个问题给出你的答案，并与同学讨论。
- 1.4. 在交付最终用户之前，或者首个版本投入使用之后，许多现代 App 程序都会有频繁的变更。为防止变更引起软件退化，请提出一些有效的解决措施。
- 1.5. 思考1.1.2节中提到的7个软件分类。请问能否将一个软件工程方法应用于所有的软件分类？并就你的答案加以解释。

扩展阅读与信息资源^①

在数千本关于计算机软件的书中，大多数讨论的是程序设计语言和软件应用系统，很少有涉及软件本身的。Pressman 和 Herron (《Software Shock》，Dorset House, 1991) 最早讨论了软件和专业开发方法的问题(针对门外汉)。Negroponte 的畅销书(《Being Digital》，Alfred A. Knopf, Inc., 1995) 提供了关于计算及其在 21 世纪的发展和影响的观点。Demarco (《Why does Software Cost So Much?》，Dorset House, 1995) 就软件和开发过程发表了一系列惊人且见解独到的论文。Ray Kurzweil (《How to Create a Mind》，Viking, 2013) 讨论了软件如何在不久的将来就会模仿人类思想，并带来人类和机器进化的“奇异性”。

Keeves (《Catching Digital》，Business Infomedia Online, 2012) 讨论了商业领导者应该如何适应以不断增大的步伐进行演化的软件。Minasi 在著作(《The Software Conspiracy: Why Software Companies Put Out Faulty Products, How They Can Hurt You, and What You Can Do》，McGraw-Hill, 2000) 中认为，现在由于软件缺陷引起的“现代灾难”将被消除并提出了解决的方法。Eubanks (《Digital Dead End: Fighting for Social Justice in the Information Age》，MIT Press, 2011) 和 Compaine (《Digital Divide: Facing a Crisis or Creating a Myth》，MIT Press, 2001) 的书认为，在 21 世纪的第一个十年里，信息(如 Web 资源)富有者和信息贫困者之间的数字鸿沟将越来越小。Kuniavsky (《Smart Things: Ubiquitous Computing User Experience Design》，Morgan Kaufman, 2010)、Greenfield (《Everyware: The Dawning Age of Ubiquitous Computing》，New Riders Publishing, 2006) 和 Loke (《Context-Aware Pervasive Systems: Architectures for a New Breed of Applications》，Auerbach, 2006) 的著作介绍了“开放世界”软件的概念，并指出在无线网络环境中软件必须能够进行适应性调整，以满足实时涌现的需求。

网上有很多讨论软件本质的信息资源，与软件过程相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

12
13

① 在每章小结之后，“扩展阅读与信息资源”一节简单介绍了本章相关资料，便于读者扩展阅读和深入理解本章内容。针对本书，我们已经建立了网站 www.mhhe.com/pressman。网站涉及软件工程的很多主题，逐章列出了相关的软件工程网站资料信息以作为本书的补充，并给出了每一本书在 Amazon.com 的链接。

软件工程

要点浏览

概念: 软件工程包括过程、一系列方法(实践)和大量工具, 专业人员借由这些来构建高质量的计算机软件。

人员: 软件工程师应用软件工程过程。

重要性: 软件工程之所以重要是因为它使得我们可以高效、高质量地构建复杂系统。它使杂乱无章的工作变得有序, 但也允许计算机软件的创建者调整其工作方式, 以更好地适应要求。

步骤: 开发计算机软件就像开发任何成功的产品一样, 需采用灵活、可适应的软

件开发过程, 完成可满足使用者要求的高质量的产品。这就是软件工程化方法。

工作产品: 从软件工程师的角度来看, 工作产品是计算机软件, 包括程序、内容(数据)和其他工作产品; 而从用户的角度来看, 工作产品是可以改善生活和工作质量的最终信息。

质量保证措施: 阅读本书的后面部分, 选择切合你所构建的软件特点的思想, 并在实际工作中加以应用。

要构建能够适应 21 世纪挑战的软件产品, 我们必须认识到以下几个简单的事实:

- 软件实际上已经深入到我们生活的各个方面, 其结果是, 对软件应用所提供的特性和功能感兴趣的人显著增多。^① 因此, 在确定软件方案之前, 需要共同努力来理解问题。
- 年复一年, 个人、企业以及政府的信息技术需求日臻复杂。过去一个人可以构建的计算机程序, 现在需要由一个庞大的团队共同实现。曾经在可预测、独立的计算环境中实现的复杂软件, 现在要将其嵌入任何产品中, 从消费性电子产品到医疗器械再到武器系统。因此, 设计已经成为关键活动。
- 个人、企业和政府在进行日常运作管理以及战略战术决策时越来越依赖于软件。软件失效会给个人和企业带来诸多不便, 甚至是灾难性的失败。因此, 软件应该具有高质量。
- 随着特定应用系统感知价值的提升, 其用户群和软件寿命也会增加。随着用户群和使用时间的增加, 其适应性和增强型性需求也会同时增加。因此, 软件需具备可维护性。

由这些简单事实可以得出一个结论: 各种形式、各个应用领域的软件

关键概念

框架活动

通用原则

原则

问题解决

SafeHome

软件工程

定义

层次

实践

软件神话

软件过程

普遍性活动

关键点 在建立解决方案之前应理解问题。

关键点 质量和可维护性都是良好设计的产物。

① 在本书的后续部分, 将这些人统称为“利益相关者”。

都需要工程化。这也是本书的主题——软件工程。

2.1 定义软件工程学科

对于软件工程,美国电气与电子工程师学会(IEEE) [IEE93a] 给出了如下定义:

软件工程是:(1)将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护,即将工程化方法应用于软件;(2)对(1)中所述方法的研究。

提问 如何定义软件工程?

然而,对于某个软件开发队伍来说可能是“系统化的、规范的、可量化的”方法,对于另外一个团队却可能是负担。因此,我们需要规范,也需要可适应性和灵活性。

软件工程是一种层次化的技术,如图 2-1 所示。任何工程方法(包括软件工程)必须构建在质量承诺的基础之上。全面质量管理、六西格玛和类似的理念^①促进了持续不断的过程改进文化,正是这种文化最终引导人们开发出更有效的软件工程方法。支持软件工程的根基在于质量关注点(quality focus)。



图 2-1 软件工程层次图

软件工程的基础是过程(process)层。软件过程将各个技术层次结合在一起,使得合理、及时地开发计算机软件成为可能。过程定义了一个框架,构建该框架是有效实施软件工程技术必不可少的。软件过程构成了软件项目管理控制的基础,建立了工作环境以便于应用技术方法、提交工作产品(模型、文档、数据、报告、表格等)、建立里程碑、保证质量及正确的管理变更。

关键点 软件工程包括过程、管理和构建软件的方法 and 工具。

15

软件工程方法(method)为构建软件提供技术上的解决方法(如何做)。方法覆盖面很广,包括沟通、需求分析、设计建模、程序构造、测试和技术支持。软件工程方法依赖于一组基本原则,这些原则涵盖了软件工程所有技术领域,包括建模活动和其他描述性技术等。

网络资源

《Cross Talk》杂志提供了关于过程、方法和工具的许多实践信息,网站地址是 www.stsc.hill.af.mil。

软件工程工具(tool)为过程和方法提供自动化或半自动化的支持。这些工具可以集成起来,使得一个工具产生的信息可被另外一个工具使用,这样就建立了软件开发的支撑系统,称为计算机辅助软件工程(computer-aided software engineering)。

提问 软件过程的元素是什么?

2.2 软件过程

软件过程是工作产品构建时所执行的一系列活动、动作和任务的集合。活动(activity)主要实现宽泛的目标(如与利益相关者进行沟通),与应用领域、项目大小、结果复杂性或者实施软件工程的重要程度没有直接关系。动作(action,如体系结构设计)包含了主要工作产品(如体系结构设计模型)生产过程中的一系列任务。任务(task)关注小而明确的目标,能够产生实际产品(如构建一个单元测试)。

引述 过程定义了活动的时间、人员、工作内容和达到预期目标的途径。

Ivar Jacobson,
Grandy Booch,
and James
Rumbaugh

^① 质量管理和相关方法的内容在本书第三部分进行讨论。

在软件工程领域，过程不是对如何构建计算机软件的严格的规定，而是一种具有可适应性的调整方法，以便于工作人员（软件团队）可以挑选适合的工作动作和任务集合。其目标通常是及时、高质量地交付软件，以满足软件项目资助方和最终用户的需求。

2.2.1 过程框架

过程框架（process framework）定义了若干个框架活动（framework activity），为实现完整的软件工程过程建立了基础。这些活动可广泛应用于所有软件开发项目，无论项目的规模和复杂性如何。此外，过程框架还包含一些适用于整个软件过程的普适性活动（umbrella activity）。一个通用的软件工程过程框架通常包含以下 5 个活动。

沟通。在技术工作开始之前，和客户（及其他利益相关者^①）的沟通与协作是极其重要的，其目的是理解利益相关者的项目目标，并收集需求以定义软件特性和功能。

策划。如果有地图，任何复杂的旅程都可以变得简单。软件项目好比是一个复杂的旅程，策划活动就是创建一个“地图”，以指导团队的项目旅程，这个地图称为软件项目计划，它定义和描述了软件工程师工作，包括需要执行的技术任务、可能的风险、资源需求、工作产品和工作进度计划。

建模。无论你是庭园设计师、桥梁建造者、航空工程师、木匠还是建筑师，你每天的工作都离不开模型。你会画一张草图来辅助理解整个项目大的构想——体系结构、不同的构件如何结合，以及其他一些特性。如果需要，可以把草图不断细化，以便更好地理解问题并找到解决方案。软件工程师也是如此，需要利用模型来更好地理解软件需求，并完成符合这些需求的软件设计。

构建。必须要对所做的设计进行构建，包括编码（手写的或者自动生成的）和测试，后者用于发现编码中的错误。

部署。软件（全部或者部分增量）交付给用户，用户对其进行评测并给出反馈意见。

上述五个通用框架活动既适用于简单小程序的开发，也可用于 WebApp 的建造以及基于计算机的大型复杂系统工程。不同的应用案例中，软件过程的细节可能差别很大，但是框架活动都是一致的。

对许多软件项目来说，随着项目的开展，框架活动可以迭代应用。也就是说，在项目的多次迭代过程中，沟通、策划、建模、构建、部署等活动不断重复。每次项目迭代都会产生一个软件增量（software increment），每个软件增量实现了部分的软件特性和功能。随着每一次增量的产生，软件将逐渐完善。

2.2.2 普适性活动

软件工程过程框架活动由很多普适性活动来补充实现。通常，这些普适性活动贯穿软件项目始终，以帮助软件团队管理和控制项目进度、质量、变更和风险。典型的普适性活动包括如下活动。

提问 五个最基本的过程框架活动是什么？

引述 爱因斯坦认为必然存在着一个对自然界的简单解释，因为上帝既不专制也不喜怒无常。然而软件工程师却无法抱侥幸心理，多数情况下，他必须面对毫无规律的复杂性。

Fred Brooks

① 利益相关者（stakeholder）就是可在项目成功中分享利益的人，包括业务经理、最终用户、软件工程师、支持人员等。Rob Thomsett 曾开玩笑说：“stakeholder 就是掌握巨额投资（stake）的人……如不照看好你的 stakeholder，就会失去投资。”

软件项目跟踪和控制——项目组根据计划来评估项目进度，并且采取必要的措施保证项目按进度计划进行。

风险管理——对可能影响项目成果或者产品质量的风险进行评估。

软件质量保证——确定和执行保证软件质量的活动。

技术评审——评估软件工程产品，尽量在错误传播到下一个活动之前发现并清除错误。

测量——定义和收集过程、项目以及产品的度量，以帮助团队在发布软件时满足利益相关者的要求。同时，测量还可与其他框架活动和普适性活动配合使用。

软件配置管理——在整个软件过程中管理变更所带来的影响。

可复用管理——定义工作产品复用的标准（包括软件构件），并且建立构件复用机制。

工作产品的准备和生产——包括生成产品（如建模、文档、日志、表格和列表等）所必需的活动。

上述每一种普适性活动都将在本书后续部分详细讨论。

关键点 普适性活动贯穿整个软件过程，主要关注项目管理、跟踪和控制。

关键点 对软件过程的适应性调整是项目成功的关键。

2.2.3 过程的适应性调整

在本节前面部分曾提到，软件工程过程并不是教条的法则，也不要求软件团队机械地执行；而应该是灵活可适应的（根据软件所需解决的问题、项目特点、开发团队和组织文化等进行适应性调整）。因此，不同项目所采用的项目过程可能有很大不同。这些不同主要体现在以下几个方面：

- 活动、动作和任务的总体流程以及相互依赖关系。
- 在每一个框架活动中，动作和任务细化的程度。
- 工作产品的定义和要求的程度。
- 质量保证活动应用的方式。
- 项目跟踪和控制活动应用的方式。
- 过程描述的详细程度和严谨程度。
- 客户和利益相关者对项目的参与程度。
- 软件团队所赋予的自主权。
- 队伍组织和角色的明确程度。

本书第1部分将详细介绍软件过程。

引述 我认为食谱只是指导方法，一个聪明的厨师每次都会变化出不同的特色。

Madame Benoit

2.3 软件工程实践

在2.2节中，曾介绍过一种由一组活动组成的通用软件过程模型，建立了软件工程实践的框架。通用的框架活动——**沟通、策划、建模、构建和部署**——和普适性活动构成了软件工程工作的体系结构的轮廓。但是软件工程的实践如何融入该框架呢？在以下几节里，读者将会对应用于这些框架活动的基本概念和原则有一个基本了解。^①

网络资源 软件工程实践方面各种深刻的想法都可以在以下网址获得：www.literateprogramming.com。

① 在本书的后面对于特定软件工程方法和普适性活动的讨论中，你应该重读本章中的相关章节。

2.3.1 实践的精髓

在现代计算机发明之前，有一本经典著作《How to Solve it》，在书中，George Polya[Pol45]列出了解决问题的精髓，这也正是软件工程实践的精髓：

1. 理解问题（沟通和分析）。
2. 策划解决方案（建模和软件设计）。
3. 实施计划（代码生成）。
4. 检查结果的正确性（测试和质量保证）。

在软件工程中，这些常识性步骤引发了一系列基本问题（与 [Pol45] 相对应）：

理解问题。虽然不愿承认，但生活中的问题很多都源于我们的傲慢。

我们只听了几秒钟就断言，好，我懂了，让我们开始解决这个问题吧。不幸的是，理解一个问题不总是那么容易，需要花一点时间回答几个简单问题：

- 谁将从问题的解决中获益？也就是说，谁是利益相关者？
- 有哪些是未知的？哪些数据、功能和特性是解决问题所必需的？
- 问题可以划分吗？是否可以描述为更小、更容易理解的问题？
- 问题可以图形化描述吗？可以建立分析模型吗？

策划解决方案。现在你理解了要解决的问题（或者你这样认为），并迫不及待地开始编码。在编码之前，稍稍慢下来做一点点设计：

- 以前曾经见过类似问题吗？在潜在的解决方案中，是否可以识别一些模式？是否已经有软件实现了所需要的数据、功能和特性？
- 类似问题是否解决过？如果是，解决方案所包含元素是否可以复用？
- 可以定义子问题吗？如果可以，子问题是否已有解决方案？
- 能用一种可以很快实现的方式来描述解决方案吗？能构建出设计模型吗？

实施计划。前面所创建的设计勾画了所要构建的系统的路线图。可能存在没有想到的路径，也可能在实施过程中会发现更好的解决路径，但是这个计划可以保证在实施过程中不至迷失方向。需要考虑的问题是：

- 解决方案和计划一致吗？源码是否可追溯到设计模型？
- 解决方案的每个组成部分是否可以证明正确？设计和代码是否经过评审？或者采用更好的方式，算法是否经过正确性证明？

检查结果。你不能保证解决方案是最完美的，但是可以保证设计足够的测试来发现尽可能多的错误。为此，需回答：

- 能否测试解决方案的每个部分？是否实现了合理的测试策略？
- 解决方案是否产生了与所要求的数据、功能和特性一致的结果？是否按照项目利益相关者的需求进行了确认？

不足为奇，上述方法大多是常识。但实际上，有充足的理由可以证明，在软件工程中采用常识将让你永远不会迷失方向。

建议 你可能会觉得Polya的方法只是简单的常识，的确如此。但是令人惊奇的是，在软件世界里，很多常识常常不为人知。

建议 理解问题最重要的是倾听。

引述 在任何问题的解决方案中都会有所发现。

George Polya

2.3.2 通用原则

原则这个词在字典里的定义是“某种思想体系所需要的重要的根本规则或者假设”。在

本书中，我们将讨论一些不同抽象层次上的原则。一些原则关注软件工程的整体，另一些原则考虑特定的、通用的框架活动（比如沟通），还有一些关注软件工程的动作（比如架构设计）或者技术任务（比如编制用例场景）。无论关注哪个层次，原则都可以帮助我们建立一种思维方式，进行扎实的软件工程实践。因此，原则非常重要。

David Hooker[Hoo96]提出了7个关注软件工程整体实践的原则，这里复述如下。^①

第1原则：存在价值

一个软件系统因能为用户提供价值而具有存在价值，所有的决策都应该基于这个思想。在确定系统需求之前，在关注系统功能之前，在决定硬件平台或者开发过程之前，问问你自己：这确实能为系统增加真正的价值吗？如果答案是不，那就坚决不做。所有的其他原则都以这条原则为基础。

第2原则：保持简洁

软件设计并不是一种随意的过程，在软件设计中需要考虑很多因素。所有的设计都应该尽可能简洁，但不是过于简化。这有助于构建更易于理解和易于维护的系统。这并不是说有些特性（甚至是内部特性）应该以“简洁”为借口而取消。的确，优雅的设计通常也是简洁的设计，但简洁也不意味着“快速和粗糙”。事实上，它经常是经过大量思考和多次工作迭代才达到的，这样做的回报是所得到的软件更易于维护且错误更少。

第3原则：保持愿景

清晰的愿景是软件项目成功的基础。没有愿景，项目将会由于它有“两种或者更多种思想”而永远不能结束；如果缺乏概念的一致性，系统就好像是由许多不协调的设计补丁、错误的集成方式强行拼凑在一起……如果不能保持软件系统体系架构的愿景，就会削弱甚至彻底破坏设计良好的系统。授权体系架构师，使其能够持有愿景，并保证系统实现始终与愿景保持一致，这对项目开发的成功至关重要。

第4原则：关注使用者

有产业实力的软件系统不是在真空中开发和使用的。通常软件系统必定是由开发者以外的人员使用、维护和编制文档等，因此必须要让别人理解你的系统。在需求说明、设计和实现过程中，牢记要让别人理解你所做的事情。对于任何一个软件产品，其工作产品都可能有很多用户。需求说明时应时刻想到用户，设计中始终想到实现，编码时想着那些要维护和扩展系统的人。一些人可能不得不对调试你所编写的代码，这使得他们成了你所编写代码的使用者，尽可能地使他们的工作简单化会大大提升系统的价值。

第5原则：面向未来

生命周期持久的系统具有更高的价值。在现今的计算环境中，需求规格说明随时会改变，硬件平台几个月后就会淘汰，软件生命周期都是以月而不是以年来衡量的。然而，真正具有“产业实力”的软件系统必须持久耐用。为了成功地做到这一点，系统必须能适应各种变化，能成功做到这一点的系统都是那些一开始就以这种路线来设计的系统。永远不要把自己的设计局限于一隅，经常问问“如果出现……应该怎样应对”，构建可以解决通用问题的

建议 在开始一个软件项目之前，应首先确保软件具有商业目标并且让用户体会到它的价值。

提问 简洁比所有巧妙的措词更加美妙。

Alexander Pope
(1688-1744)

21

关键点 如果软件有价值，那么其价值在其生命周期中将发生变更。因此，软件必须构建成可维护的。

^① 这里的引用得到了作者的授权 [Hoo96]. Hooker 定义这些原则的模式请参见：<http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment>。

系统,为各种可能的方案做好准备,^①这很可能会提高整个系统的可复用性。

第6原则:提前计划复用

复用既省时又省力^②。软件系统开发过程中,高水平的复用是很难实现的一个目标。曾有人宣称代码和设计复用是面向对象技术带来的主要好处,然而,这种投入的回报不会自动实现。为达到面向对象(或是传统)程序设计技术所能够提供的复用性,需要有前瞻性的设计和计划。系统开发过程中的各个层面上都有多种技术可实现复用……提前做好复用计划将降低开发费用,并增加可复用构件以及构件化系统的价值。

第7原则:认真思考

这最后一条规则可能是最容易被忽略的。在行动之前清晰定位、完整思考通常能产生更好的结果。仔细思考可以提高做好事情的可能性,而且也能获得更多的知识,明确如何把事情做好。如果仔细思考过后还是把事情做错了,那么,这就变成了很有价值的经验。思考就是学习和了解本来一无所知的事情,使其成为研究答案的起点。把明确的思想应用在系统中,就产生了价值。使用前六条原则需要认真思考,这将带来巨大的潜在回报。

如果每位工程师、每个开发团队都能遵从 Hooker 这七条简单的原则,那么,开发复杂计算机软件系统时所遇到的许多困难都可以迎刃而解。

2.4 软件开发神话

软件开发神话,即关于软件及其开发过程的一些被人盲目相信的说法,这可以追溯到计算技术发展的初期。神话具有一些特点,让人觉得不可捉摸。例如,神话看起来是事实的合理描述(有时确实包含真实的成分),它们符合直觉,并且经常被那些知根知底的有经验的从业人员拿来宣传。

今天,大多数有见地的软件工程师已经意识到软件神话的本质——它实际上误导了管理者和从业人员对软件开发的态​​度,从而引发了严重的问题。然而,由于习惯和态​​度的根深蒂固,软件神话遗风犹在。

管理神话。像所有领域的经理一样,承担软件职责的项目经理肩负着维持预算、保证进度和提高质量的压力。就像溺水人抓住稻草一样,软件经理经常依赖软件神话中的信条,只要它能够减轻以上的压力(即使是暂时性的)。

网络资源 软件项目经理网有助于人们澄清这些神话: www.spmn.com。

神话:我们已经有了​​一本写满软件开发标准和规程的宝典,难道不能提供我们所需要了解的所有信息吗?

事实:这本宝典也许的确已经存在,但它是否在实际中采用了?从业人员是否知道这本书的存在呢?它是否反映了软件工程的现状?是否全面?是否可以适应不同的应用环境?是否在缩短交付时间的同时还关注产品质量的保证?在很多情况下,问题的答案是否定的。

神话:如果我们未能按时完成计划,我们可以通过增加程序员人数而赶上进度(即所谓的“蒙古游牧”概念)。

事实:软件开发并不是像机器制造那样的机械过程。Brooks 曾说过 [Bro95]:“在软件工

① 把这个建议发挥到极致会非常危险,设计通用方案会带来性能损失,并降低特定的解决方案的效率。

② 尽管对于准备在未来的项目中复用软件的人而言,这种说法是正确的,但对于设计和实现可复用构件的人来说,复用的代价会很昂贵。研究表明,设计和开发可复用构件比直接开发目标软件要增加 25% ~ 200% 的成本,在有些情况下,这些费用差别很难核实。

程中,为赶进度而增加人手只能使进度更加延误。”初看,这种说法似乎与直觉不符。然而,当新人加入到一个软件项目后,原有的开发人员必须要牺牲本来的开发时间对后来者进行培训,因此减少了本应用于高效开发的时间。只有在有计划且有序进行的情况下,增加人员对项目进度才有意义。

神话:如果决定将软件外包给第三方公司,就可以放手不管,完全交给第三方公司开发。

事实:如果开发团队不了解如何在内部管理和控制软件项目,那么将无一例外地在外包项目中遇到困难。

客户神话.软件产品的客户可能是隔壁的某个人、楼下的一个技术团队、市场/销售部门或者签订软件合同的某个外部公司。多数情况下,客户之所以相信所谓的软件神话,是因为项目经理和从业人员没有及时纠正他们的错误信息。软件神话导致客户错误的期望,最终导致对开发者的不满。

神话:有了对项目目标的大概了解,便足以开始编写程序,可以在之后的项目开发过程中逐步充实细节。

事实:虽然通常很难得到综合全面且稳定不变的需求描述,但是对项目目标模糊不清的描述将为项目实施带来灾难。要得到清晰的需求描述(经常是逐步变得清晰的),只能通过客户和开发人员之间的持续有效的沟通。

神话:虽然软件需求不断变更,但是因为软件是弹性的,因此可以很容易地适应变更。

事实:软件需求的确在随时变更,但随变更引入的时机不同,变更所造成的影响也不同。如果需求变更提出得较早(比如在设计或者代码开发之前),则对费用的影响较小^①;但是,随着时间的推移,变更的代价也迅速增加——因为资源已经被分配,设计框架已经建立,而变更可能会引起的剧变,需要添加额外的资源或者修改主要设计。

从业者神话.在60多年的编程文化的滋养下,软件开发人员依然深信着各种神话。在软件业发展早期,编程被视为一种艺术。旧有的方式和态度根深蒂固。

神话:当我们完成程序并将其交付使用之后,我们的任务就完成了。

事实:曾经有人说过,对于编程来说,开始得越早,耗费的时间就越长。业界的一些数据显示,60%~80%的工作耗费在软件首次交付顾客使用之后。

神话:直到程序开始运行,才能评估其质量。

事实:最有效的软件质量保证机制之一——技术评审,可以从项目启动就开始实行。软件评审(第20章)作为“质量过滤器”,已经证明其可以比软件测试更为有效地发现多种类型的软件缺陷。

神话:对于一个成功的软件项目,可执行程序是唯一可交付的工作成果。

事实:软件配置包括很多内容,可执行程序只是其中之一。各种工作产品(如模型、文档、计划)是成功实施软件工程的基础,更重要的是,为软件技术支持提供了指导。

建议 在项目开始之前,尽可能努力了解工作内容。也许难以明确所有细节,但你了解得越多,所面临的风险就越低。

24

建议 每当你认为没有时间采用软件工程方法时,就再问问自己:“是否有时间重做整个软件?”

① 许多软件工程师采纳了“敏捷”(agile)开发方法,通过增量的方式逐步纳入变更,以便控制变更的影响范围和成本。本书第5章讨论敏捷方法。

神话：软件工程将导致我们产生大量无用文档，并因此降低工作效率。

事实：软件工程并非以创建文档为目的，而是为了保证软件产品的开发质量。好的质量可以减少返工，从而加快交付时间。

目前，大多数软件专业人员已经认识到软件神话的谬误。对于软件开发真实情况的正确理解是系统阐述如何使用软件工程方法解决实际问题的第一步。

25

2.5 这一切是如何开始的

每个软件工程项目都来自业务需求——对现有应用程序缺陷的纠正，改变遗留系统以适应新的业务环境，扩展现有应用程序功能和特性，或者开发某种新的产品、服务或系统。

在软件项目的初期，业务需求通常是在简短的谈话过程中非正式地表达出来的。以下这段简短谈话就是一个典型的例子。

SafeHome[⊖] 如何开始一个软件项目

【场景】 CPI 公司的会议室里。CPI 是一个虚构的为家庭和贸易应用生产消费产品的公司。

【人物】 Mal Golden, 产品开发部高级经理；Lisa Perez, 营销经理；Lee Warren, 工程经理；Joe Camalleri, 业务发展部执行副总裁。

【对话】

Joe: Lee, 我听说你们那帮家伙正在开发一个产品——通用的无线盒？

Lee: 哦，是的，那是一个很棒的产品，只有火柴盒大小。我们可以把它放在各种传感器上，比如数码相机，总之任何东西里。采用 802.11n 无线网络协议，可以通过无线连接获得它的输出。我们认为它可以带来全新的一代产品。

Joe: Mal, 你觉得怎么样呢？

Mal: 我当然同意。事实上，随着这一年来销售业绩的趋缓，我们需要一些新的产品。Lisa 和我已经做了一些市场调查，我们都认为该系列产品具有很大的市场潜力。

Joe: 多大，底线是多少？

Mal (避免直接承诺)：Lisa, 和他谈谈我们的想法。

Lisa: 这是新一代的家庭管理产品，我们称之为“SafeHome”。产品采用新型无线接口，给家庭和小型商务从业人士提供一个由电脑控制的系统——住宅安全、监视，仪表和设备控制。例如，你可以在回家的路上关闭家里的空调，或者如此这类的应用。

Lee (插话)：Joe, 工程部已经作了相关的技术可行性研究。它可行且制造成本不高。大多数硬件可以在市场购买产品，不过软件方面是个问题，但也不是我们不能做的。

Joe: 有意思！我想知道底线。

Mal: 在美国，70% 的家庭拥有电脑。如果我们定价合适，这将成为一个十分成功的产品。到目前为止，只有我们拥有这一无线控制盒技术。我们将在这方面保持两年的领先地位。收入吗，在第二年大约可达到 3000 万到 4000 万。

Joe (微笑)：我很感兴趣，让我们继续讨论一下。

⊖ SafeHome 项目将作为一个案例贯穿本书，以便说明项目组在开发软件产品过程中的内部工作方式。公司、项目和人员都是虚构的，但场景和问题是真实的。

除了一带而过地谈到软件，这段谈话中几乎没有提及软件开发项目。然而，软件将是 SafeHome 产品线成败的关键。只有 SafeHome 软件成功，该产品才能成功。只有嵌入其中的软件产品满足顾客的需求（尽管还未明确说明），产品才能被市场所接受。我们将在后面的几章中继续讨论 SafeHome 中软件工程的话题。

26

2.6 小结

软件工程包含过程、方法和工具，这些工具使得快速构建高质量的复杂计算机系统成为可能。软件过程包括五个框架活动：沟通、策划、建模、构建和部署，这些活动适用于所有软件项目。软件工程实践遵照一组核心原则，是一项解决问题的活动。

尽管我们关于构建软件所需的软件知识和技能增长了，但仍有大量的软件神话将管理者和从业人员诱入歧途。随着对软件工程理解的深化，你就会逐渐明白，为什么无论何时遇到这些神话，都要不遗余力地揭露。

习题与思考题

- 2.1 图 2-1 中，将软件工程的三个层次放在了“质量关注点”这层之上。这意味着在整个开发组织内采用质量管理活动，如“全面质量管理”。仔细研究并列全面质量管理活动中关键原则的大纲。
- 2.2 软件工程对构建 WebApp 是否适用？如果适用，需要如何改进以适应 WebApp 的独特特点？
- 2.3 随着软件的普及，由于程序错误所带来的公众风险已经成为一个愈加重要的问题。设想一个真实场景：由于软件错误而引起“世界末日”般的重大危害（危害社会经济或是人类生命财产安全）。
- 2.4 用自己的话描述过程框架。当我们谈到框架活动适用于所有的项目时，是否意味着对于不同规模和复杂度的项目可应用相同的工作任务？请解释。
- 2.5 普适性活动存在于整个软件过程中，你认为它们均匀分布于软件过程中，还是集中在某个或者某些框架活动中？
- 2.6 在 2.4 节所列举的神话中，增加两种软件神话，同时指出与其相对应的真实情况。

扩展阅读与信息资源

软件工程及软件过程的当前发展状况可以参阅一些期刊，如《IEEE Software》《IEEE Computer》《CrossTalk》和《IEEE Transactions on Software Engineering》。《Application Development Trends》和《Cutter IT Journal》等行业期刊通常包含一些关于软件工程的文章。每年，IEEE 和 ACM 资助的研讨会论文集《Proceeding of the International Conference on Software Engineering》都是对当年学术成果的总结，并且在《ACM Transactions on Software Engineering and Methodology》《ACM Software Engineering Notes》和《Annals of Software Engineering》等期刊上有进一步深入讨论。当然，在互联网上有很多关于软件工程和软件过程的网页。

27

近年出版了许多关于软件过程和软件工程的书籍，有些是关于整个过程的概要介绍，有些则深入讨论过程中一些重要专题。下面是一些畅销书（除本书之外）：

《SWEBOK: Guide to the Software Engineering Body of Knowledge》^①，IEEE，2013，见 <http://www.computer.org/portal/web/swebok>。

Andersson, E. 等，《Software Engineering for Internet Applications》，MIT Press，2006。

Braude, E. 和 M. Bernstein，《Software Engineering: Modern Approaches》，2nd ed., Wiley,

① 可从 <http://www.computer.org/portal/web/swebok/htmlformat> 免费下载。

2010。

Christensen, M. 和 R. Thayer, 《A Project Manager's Guide to Software Engineering Best Practices》, IEEE-CS Press (Wiley), 2002。

Glass, R., 《Fact and Fallacies of Software Engineering》, Addison-Wesley, 2002。

Hussain, S., 《Software Engineering》, I K International Publishing House, 2013。

Jacobson, I., 《Object-Oriented Software Engineering: A Use Case Driven Approach》, 2nd ed., Addison-Wesley, 2008。

Jalote, P., 《An Integrated Approach to Software Engineering》, 3rd ed., Springer, 2010。

Pfleeger, S., 《Software Engineering: Theory and Practice》, 4th ed., Prentice Hall, 2009。

Schach, S., 《Object-Oriented and Classical Software Engineering》, 8th ed., McGraw-Hill, 2010)。

Sommerville, I., 《Software Engineering》, 9th ed., Addison-Wesley, 2010。

Stober, T., 和 U. Hansmann, 《Agile Software Development: Best Practices for Large Development Projects》, Springer, 2009。

Tsui, F., 和 O.karam, 《Essentials of Software Engineering》, 2nd ed., Jones&Bartlett Publishers, 2009。

Nygard(《Release it!: Design and Deploy Production-Ready Software》, Pragmatic Bookshelf, 2007)、Richardson 和 Gwaltney (《Ship it! A Practical Guide to Successful Software Projects》, Pragmatic Bookshelf, 2005) 以及 Humble 和 Farley (《Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation》, Addison-Wesley, 2010) 的书给出了大量有用的指导原则, 可用于部署活动。

在过去的几十年里, IEEE、ISO 以及附属其下的标准化组织发布了大量软件工程标准。Moore (《The Road Map to Software Engineering: A Standards-Based Guide》, IEEE Computer Society Press[Wiley], 2006) 对相关标准进行了调查并指出了这些标准应如何应用到实际工程中。

网上有很多有关软件工程和软件过程相关问题的信息资源, 与软件过程相关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

软件过程

本部分将介绍软件过程，它提供了软件工程实践的框架。在本部分的各章中将涉及以下问题：

- 什么是软件过程？
- 软件过程中，有哪些通用的框架活动？
- 如何建立过程模型？什么是过程模式？
- 什么是惯用过程模型？有哪些优缺点？
- 为什么现代软件工程关注敏捷问题？
- 什么是敏捷软件开发？它与传统的过程模型有什么区别？

在解决了上述问题之后，就可以对软件工程实践的应用背景有更清楚的认识。

软件过程结构

要点浏览

概念: 在开发产品或构建系统时, 遵循一系列可预测的步骤(即路线图)是非常重要的, 它有助于及时交付高质量的产品。软件开发中所遵循的路线图就称为“软件过程”。

人员: 软件工程师及其管理人员根据需要调整开发过程, 并遵循该过程。除此之外, 软件的需求方也需要参与过程的定义、建立和测试。

重要性: 软件过程提高了软件工程活动的稳定性、可控性和有组织性, 如果不进行控制, 软件活动将变得混乱。但是, 现代软件工程方法必须是“灵活”的, 也就是要求软件工程活动、控制以及工作

产品适合于项目团队和将要开发的产品。

步骤: 具体来讲, 采用的过程依赖于构造软件的特点。飞机航空系统的软件与网站的建设可能需要采用两种截然不同的软件过程。

工作产品: 从软件工程师的角度来看, 工作产品体现为在执行过程所定义的任务和活动的过程中, 所产生的程序、文档和数据。

质量保证措施: 有大量的软件过程评估机制, 开发机构可以评估其软件过程的“成熟度”。然而, 表征软件过程有效性的最好指标还是所构建产品的质量、及时性和寿命。

Howard Baetjer Jr.[Bae98] 曾著书从经济学家的角度分析软件和软件工程, 该书引人入胜, 对软件过程评述如下:

软件同其他资产一样, 是知识的具体体现, 而知识最初都是以分散的、不明确的、隐蔽的且不完整的形式广泛存在的, 因此, 软件开发是一个社会学习的过程。软件过程是一个对话的过程, 在对话中, 获取需要转化为软件的知识, 并在软件中实现这些知识。软件过程提供了用户与设计人员之间、用户与不断演化的工具之间以及设计人员与不断演化的工具(技术)之间的互动。软件开发是一个迭代的过程, 在其中演化的工具本身就作为沟通的媒介, 任何新一轮对话都可以从参与的人员中获得更有用的知识。

构建计算机软件确实是一个迭代的社会学习的过程, 其输出——即 Baetjer 所称的“软件资产”——是知识的载体, 这些知识在过程执行中进行收集、提炼和组织。

但从技术的角度如何确切地定义软件过程呢? 本书将软件过程定义为一个为创建高质量软件所需要完成的活动、动作和任务的框架。过程与软件工程同义吗? 答案是“是, 也不是”。软件过程定义了软件工程化中采用的方法, 但软件工程还包含该过程中应用的技术——技术方法和自动化工具。

更重要的是, 软件工程是由有创造力、有知识的人完成的, 他们根据产品构建的需要和

关键概念

- 通用过程模型
- 过程评估
- 过程流
- 过程改进
- 过程模式
- 任务集

市场需求来选取成熟的软件过程。

3.1 通用过程模型

在第2章中，过程定义为在工作产品构建过程中所需完成的工作活动、动作和任务的集合。这些活动、动作、任务中的每一个都隶属于某一框架或者模型，框架或模型定义了它们与过程之间或者相互之间的关系。

软件过程示意图如图3-1所示。由图可以看出，每个框架活动由一系列软件工程动作构成；每个软件工程动作由任务集来定义，这个任务集明确了将要完成的工作任务、将要产生的工作产品、所需要的质量保证点，以及用于表明过程状态的里程碑。

关键点 在软件过程中，技术工作的层次包括活动，活动由动作构成，动作由任务构成。

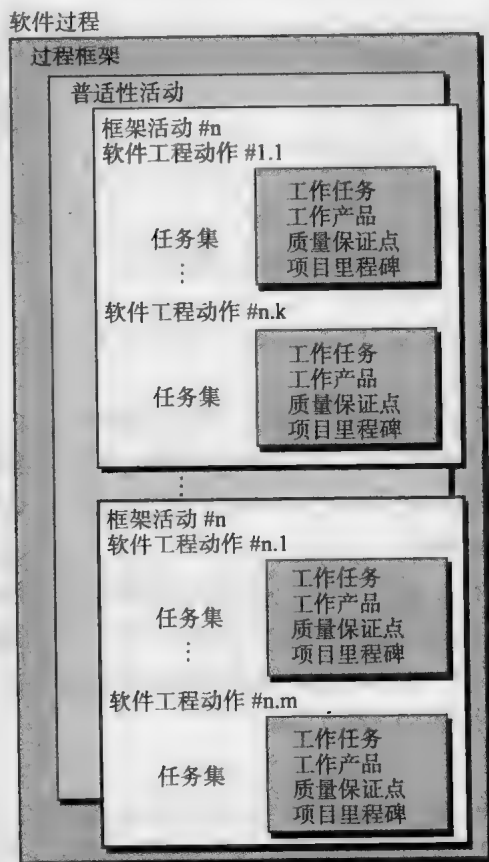


图 3-1 软件过程框架

正如在第2章中讨论的，软件工程的通用过程框架定义了五种框架活动——沟通、策划、建模、构建以及部署。此外，一系列普适性活动——项目跟踪控制、风险管理、质量保证、配置管理、技术评审以及其他活动——贯穿软件过程始终。

你也许注意到了，软件过程的一个很重要的方面还没有讨论，即过程流（process flow）。过程流描述了在执行顺序和执行时间上如何组织框架中的活动、动作和任务，如图3-2所示。

提问 什么是过程流？

线性过程流（linear process flow）从沟通到部署顺序执行五个框架活动（参见图3-2a）。迭代过程流（iterative process flow）在执行下一个活动前重复执行之前的一个或多个活动

(参见图 3-2b)。演化过程流 (evolutionary process flow) 采用循环的方式执行各个活动, 每次循环都能产生更为完善的软件版本 (参见图 3-2c)。并行过程流 (parallel process flow) (参见图 3-2d) 将一个或多个活动与其他活动并行执行 (例如, 软件一个方面的建模可以同软件另一个方面的构建活动并行执行)。

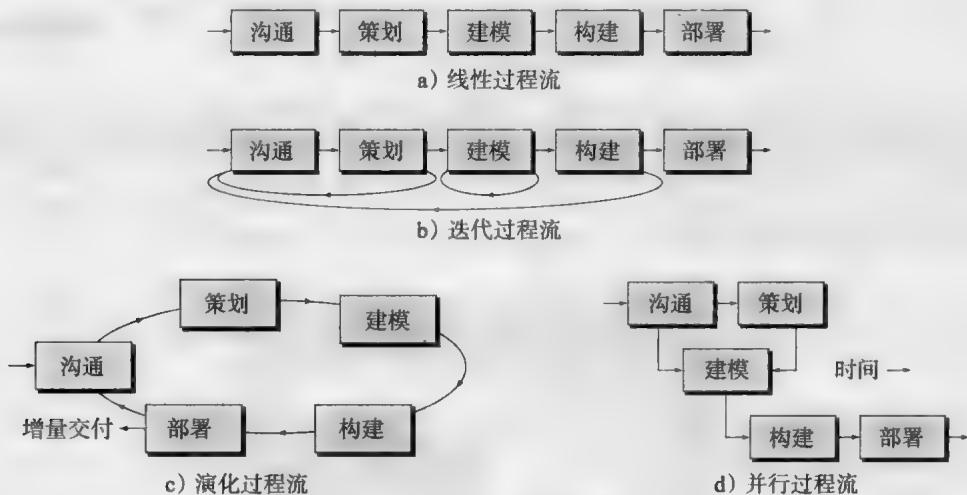


图 3-2 过程流

3.2 定义框架活动

尽管第 2 章描述了 5 种框架活动, 并给出了每种活动的基本定义, 但是软件团队要在软件过程中具体执行这些活动中的任何一个, 还需要更多信息。因此, 我们面临一个关键问题: 针对给定的问题、开发人员和利益相关者, 哪些动作适合于框架活动?

对于由个人负责的小型软件项目 (可能远程), 其需求简单明确, 沟通活动也许仅仅是与合适的利益相关者的一个电话或一封邮件。因此, 主要的动作是电话交流, 这个动作所包括的主要工作任务 (任务集) 有:

1. 通过电话与利益相关者取得联系。
2. 讨论需求并做记录。
3. 将笔记整理成一份简单的书面需求。
4. 通过 E-mail 请利益相关者审阅并认可。

如果项目有多个利益相关者, 则要复杂得多, 每个参与人员都有着不同的需求 (有时这些需求甚至是相互冲突的), 沟通活动可能会包含 6 个不同的动作 (具体参见第 8 章): 起始、需求获取、需求细化、协商、规格说明和确认。每个软件工程动作都可能有很多工作任务和一些不同的工作产品。

3.3 明确任务集

我们再来看图 3-1, 每一个软件工程动作 (如需求获取以及与沟通活动相关的动作) 都由若干个任务集 (task set) 构成, 而每一个任务集都由软件工作任务、相关工作产品、质量保证点和项目里程碑组成。需要选择最能满足项目需要并适合开发团队特点的任

引述 如果过程正确, 就会得到应得的结果。

Takashi Osada

提问 框架活动如何随着项目性质的变化而变化?

关键点 不同的项目需要不同的任务集。软件团队根据问题和项目的特点选择任务集。

务集。这就意味着软件工程师可以根据软件项目的特定需要和项目团队的特点做适当的调整。

信息栏 任务集

任务集定义了为达到一个软件工程师动作的目标所需要完成的工作。例如，需求获取（通常称为“需求收集”）就是发生在沟通活动中的一个重要的软件工程师动作。需求获取的目的是理解利益相关者对将构建的软件的需求。

对于一个小型、相对简单的项目而言，需求获取的任务集可能包括：

1. 制定项目的利益相关者列表。
2. 邀请所有的利益相关者参加一个非正式会议。
3. 征询每个人对于软件特性和功能的需求。
4. 讨论需求，并确定最终的需求列表。
5. 划定需求优先级。
6. 标出不确定域。

对于大型、复杂的软件工程项目而言，可能需要如下不同的任务集：

1. 制定项目的利益相关者列表。
2. 和利益相关者的每个成员分别单独讨论，获取所有的要求。

3. 基于利益相关者的输入，建立初步的功能和特性列表。

4. 安排一系列促进需求获取的会议。

5. 组织会议。

6. 在每次会议上建立非正式的用户场景。

7. 根据利益相关者的反馈，进一步细化用户场景。

8. 建立一个修正的利益相关者需求列表。

9. 使用质量功能部署技术，划分需求优先级。

10. 将需求打包以便于软件可以实施增量交付。

11. 标注系统的约束和限制。

12. 讨论系统验证方法。

上面两种任务集都可以完成需求获取，但是无论从深度还是形式化的程度上来说，二者都有很大区别。软件团队采取适当的任务集以达到每个动作的目的，并且保持软件质量和开发的敏捷性。

34

3.4 过程模式

每个软件团队在软件过程里都会遇到很多问题。针对这些问题，如果软件团队能够得到已有的经过验证的解决方案，将有助于他们快速地分析和解决问题。过程模式（process pattern）^①描述了软件工程师工作中遇到的过程相关的问题，明确了问题环境并给出了针对该问题的一种或几种可证明的解决方案。通俗地讲，过程模式提供了一个模板 [Amb98]——一种在软件过程的背景下统一描述问题解决的方法。通过模式组合，软件团队可以解决问题并定义最符合项目需求的开发过程。

我们可以在不同抽象层次上定义模式^②。在某些情况下，模式可以描述一个与完整过程模型（例如原型开发）相关的问题（及其解决方案）；在其他的情况下，模式可以描述一个与

提问 什么是过程模式？

① 关于模式的详细讨论参见第11章。

② 模式的概念广泛应用于软件工程的活动中。第11、13、15、16和20章将分别讨论分析模式、设计模式和测试模式。本书第四部分将讨论项目管理活动中的模式和反模式。

框架活动（如策划）或者框架活动中的一动作（如项目估算）相关的问题（及其解决方案）。

Ambler[Amb98]提出了下面的过程模式的描述模板：

模式名称。模式名称应能清楚地表述该模式在软件过程中的含义（例如技术评审）。

驱动力。模式的使用环境及主要问题，这些问题会显现在软件过程中并可能影响解决方案。

类型。定义模式类型。Ambler[Amb98]提出了三种类型：

1. **步骤模式（stage pattern）**——定义了与过程的框架活动相关的问题。由于框架活动包括很多动作和工作任务，因此步骤模式包括与步骤（框架活动）有关的许多任务模式（见以下描述）。例如，建立沟通可能作为一个步骤模式，该步骤模式可能包括需求收集等任务模式。
2. **任务模式（task pattern）**——定义了与软件工程动作或是工作任务相关、关系软件工程实践成败的问题（例如，需求收集是一个任务模式）。
3. **阶段模式（phase pattern）**——定义在过程中发生的框架活动序列，即使这些活动流本质上是迭代的。例如，螺旋模型和原型开发^①就可能是两种阶段模式。

启动条件。它描述的是模式应用的前提条件。在应用模式之前需要明确：（1）在此之前，整个开发组织或是开发团队内已经有哪些活动？（2）过程的进入状态是什么？（3）已经有哪些软件工程信息或是项目信息？

例如，策划模式（阶段模式）需要的前提条件有：（1）客户和软件工程师已经建立了合作的交流机制；（2）已经成功完成一些客户沟通模式中特定的任务模式；（3）项目范围、基本业务需求和项目限制条件已经确定。

问题。描述模式将要解决的具体问题。

解决方案。描述如何成功实现模式。这部分主要讨论随着模式的启动，过程的初始状态（模式应用之前就已经存在）是如何发生改变的。解决方案也描述了随着模式的成功执行，模式启动之前所获得的软件工程信息和项目信息是如何变换的。

结果。描述模式成功执行之后的结果。模式完成时需要明确：（1）必须完成哪些开发组织或是开发团队相关的活动？（2）过程的结束状态是什么？（3）产生了哪些软件工程信息或是项目信息？

相关模式。以层次化或其他图的方式列举与该模式相关的其他过程模式。例如步骤模式沟通包括了一组任务模式：项目团队组织、合作指导原则定义、范围分解、需求收集、约束描述以及场景模式的创建等。

已知应用和实例。说明该模式可应用的具体实例。例如，沟通在每一个软件项目的开始都是必需的，建议在整个软件项目过程中采用，并规定在部署活动中必须进行。

过程模式提供了一种有效的机制，用以解决任何与软件过程相关的问题。模式使得软件工程组织能够从高层抽象开始（阶段模式）建立层次化的过程描述。高层抽象描述又进一步细化为一系列步骤模式以描述框架活动，然后每一个步骤模式又进一步逐层细化为更详细的任务模式。过程模

引述 模式的重复与软件模块的重复完全不同。事实上，不同的模块是独特的，而模式却是相同的。

Christopher
Alexander

关键点 模式模板提供了描述模式的一般性方法。

引述 我们认为软件开发人员遗漏了一个事实：多数机构并不清楚他们在做什么，他们以为清楚了，但实际上不清楚。

Tom DeMarco

网络资源 可在下面的网站查看过程模式的全面资源：www.amblysoft.com/processPatternsPage.html。

① 第4章将讨论这些阶段模式。

式一旦建立起来,就可以进行复用以定义各种过程变体,即软件开发团队可以将模式作为过程模型的构建模块,定制特定的过程模型。

信息栏 过程模式实例

当利益相关者对工作成果有大致的想法,但对具体的软件需求还不确定时,下述简化的过程模式描述了可采用的方法。

模式名称。需求不清。

目的。该模式描述了一种构建模型(或是原型系统)的方法,使得利益相关者可以反复评估,以便识别和确定软件需求。

类型。阶段模式。

启动条件。在模式启动之前必须满足以下四个条件:(1)确定利益相关者;(2)已经建立起利益相关者和软件开发团队之间的沟通方式;(3)利益相关者确定了需要解决的主要问题;(4)对项目范围、基本业务需求和项目约束条件有了初步了解。

问题。需求模糊或者不存在,但都清楚地认识到项目存在问题,且该问题需要

通过软件解决。利益相关者不确定他们想要什么,即他们无法详细描述软件需求。

解决方案。描述了原型开发过程,详见4.1.3节。

结果。开发了软件原型,识别了基本的需求(例如交互模式、计算特性、处理功能等),并获得了利益相关者的认可。随后,可能有两种结果:(1)原型系统可以通过一系列的增量开发,演化成为软件产品;(2)原型系统被抛弃,采用其他过程模式建立了产品软件。

相关模式。以下模式与该模式相关:客户沟通,迭代设计,迭代开发,客户评价,需求抽取。

已知应用和实例。当需求不确定时,推荐原型开发方法。

3.5 过程评估与改进

软件过程并不能保证软件按期交付,也不能保证软件满足客户要求,或是软件具备了长期质量保证的技术特点(第19章)。软件过程模型必须与切实的软件工程实践相结合(本书第二部分)。另外,对过程本身也要进行评估,以确保满足了成功软件工程所必需的基本过程标准要求^①。

在过去的几十年中,人们提出了很多种不同的软件过程评估和改进方法。

用于过程改进的CMMI[®]标准评估方法(Standard CMMI Assessment Method for Process Improvement, SCAMPI)——提供了五步的过程评估模型:启动(initiating)、诊断(diagnosing)、建立(establishing)、执行(acting)和学习(learning)。SCAMPI方法采用SEI的CMMI作为评估的依据[SEI00]。

用于组织内部过程改进的CMM评估(CMM-Based Appraisal for Internal Process Improvement, CBA IPI)——采用SEI的CMM作为评估的依据[Dun01],提供了一种诊断方法,用以分析软件开发机构的相对成熟度。

SPICE(ISO/IEC 15504)——该标准定义了软件过程评估的一系列要求。该标准的目的是帮助软件开发组织建立客观的评价体系,以评估定义

关键点 以改进为目标,评估力求理解软件过程的当前状态。

引述 软件组织还存在很大的能力缺陷,难以将其从整个项目中所获得的经验转化成资产。

NASA

① SEI CMMI[CMM07]丰富详实地介绍了软件过程的基本特征,以及过程成功的标准。

② 有关CMMI的详细介绍参见37.3节。

的软件过程的有效性 [ISO08]。

软件 ISO 9001:2000——这是一个通用标准，任何开发组织如果希望提高所提供的产品、系统或服务的整体质量，都可以采用这个标准。因此，该标准可直接应用于软件组织和公司 [Ant06]。

有关软件评估和过程改进方法的详细讨论参见第 37 章。

3.6 小结

一个软件工程通用过程模型包含了一系列的框架和普适性活动、动作以及工作任务。每一种不同的过程模型都可以用不同的过程流来描述， workflow 描述了框架活动、动作和任务是如何按顺序组织的。过程模式用来解决软件过程中遇到的共性问题。

习题与思考题

- 3.1 在本章的介绍中，Baetjer 说过：“软件过程提供了用户与设计人员之间、用户与开发工具之间以及设计人员与开发工具之间的互动。”对以下四个方面各设计五个问题：（1）设计人员应该问用户的；（2）用户应该问设计人员的；（3）用户对将要构建的软件的自问；（4）设计人员对于软件产品和建造该产品采取的软件过程的自问。
- 3.2 讨论 3.1 节所描述的不同过程流之间的区别。你是否能够确定适用于所描述的每种通用流的问题类型？
- 3.3 为沟通活动设计一系列动作，选定一个动作作为其设计一个任务集。
- 3.4 在沟通过程中，遇到两位对软件如何做有着不同想法的利益相关者是很常见的问题。也就是说，你得到了相互冲突的需求。设计一种过程模式（可以是步骤模式），利用 3.4 节中针对此类问题的模板，给出一种行之有效的解决方法。

扩展阅读与信息资源

大多数软件工程课本都会详细介绍过程模型。Sommerville（《Software Engineering》，9th ed., Addison-Wesley, 2010）、Schach（《Object-Oriented and Classical Software Engineering》，8th ed., McGraw-Hill, 2010）以及 Pfleeger 和 Atlee（《Software Engineering: Theory and Practice》，4th ed., Prentice Hall, 2009）的书中介绍了这些传统的模型，并讨论了它们的优点和缺点。Munch 和他的同事（《Software Process Definition and Management》，Springer, 2012）介绍了过程和产品的软件和系统工程观点。Glass（《Facts and Fallacies of Software Engineering》，Prentice-Hall, 2002）提出了一种保证软件工程过程的不加修饰且实用的观点。Brooks（《The Mythical Man-Month》，2d ed., Addison-Wesley, 1995）在他的书中虽然没有直接讲过程，但是他用一生的项目学识讲了和过程相关的每一个方面。

Firesmith 和 Henderson-Sellers（《The OPEN Process Framework: An introduction》，Addison-Wesley, 2001）为创建“灵活但有序的软件过程”提出了一个通用的模板，并讨论了过程属性和目的。Madachy（《Software Process Dynamics》，Wiley-IEEE, 2008）讨论了一种对软件过程中的相关技术和社会因素进行分析的建模技术。Sharpe 和 McDermott（《Workflow Modeling: Tools for Process Improvement and Application Development》，2nd ed., Artech House, 2008）介绍了为软件和商业过程建模的工具。

网上有大量关于软件工程和软件过程的信息，和软件过程有关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

过程模型

要点浏览

概念：过程模型为软件工作提供了特定的路线图，该路线图规定了所有活动的流程、动作、任务、迭代的程度、工作产品及要完成的工作应如何组织。

人员：软件工程师及其管理人员根据他们的要求采用一种过程模型，并遵循该过程模型。此外，软件的需求方也需要参与过程的定义、构建和测试。

重要性：软件过程提高了软件工程活动的稳定性、可控性和有组织性，如果没有过程约束，软件活动将失控并变得混乱。但是，现代软件工程方法必须是“灵活的”，也就是要求软件工程活动、控制以

及工作产品适合于项目团队和将要开发的产品。

步骤：过程模型为软件人员提供了开展软件工作需要遵循的步骤。

工作产品：从软件工程师的角度来看，工作产品是对过程所定义的活动和任务的格式化描述。

质量保证措施：有大量的软件过程评估机制，开发机构可以评估其软件过程的“成熟度”。然而，表征软件过程有效性的最好指标还是所构建产品的质量、及时性和寿命。

最早提出过程模型是为了改变软件开发的混乱状况，使软件开发更加有序。历史证明，这些模型为软件工作提出了大量有用的结构，并为软件团队提供了有效的路线图。尽管如此，软件工作及其产品仍然停留在“混乱的边缘”。

在一篇探讨软件世界中有序和混乱之间奇怪关系的论文中，Nogueira和他的同事指出 [Nog00]：

混乱的边缘可定义为“有序和混乱之间的一种自然状态，结构化和反常之间的重大妥协” [Kau95]。混乱的边缘可以被视为一种不稳定和部分结构化的状态……它的不稳定是因为它不停地受到混乱或者完全有序的影响。

我们通常认为有序是自然的完美状态。这可能是个误区……研究证实，打破平衡的活动会产生创造力、自我组织的过程和更高的回报 [Roo96]。完全的有序意味着缺乏可变性，而可变性在某些不可预测的环境下往往是一种优势。变更通常发生在某些结构中，这些结构使得变更可以被有效组织，但还不是死板得使得变更无法发生。另一方面，太多的混乱会使协调和一致成为不可能。缺少结构并不意味着无序。

上面这段话的哲学思想对于软件工程有着重要的意义。本章所描述的每个过程模型都试图在找出混乱世界中的秩序和适应不断发生的变化这两种要求之间寻求平衡。

关键概念

面向方面的软件开发
基于构件的开发
并发模型
演化过程模型
形式化方法模型
增量过程模型
个人软件过程
过程建模工具
过程技术
原型开发
螺旋模型
团队软件过程
统一过程
V模型
瀑布模型

4.1 惯用过程模型

惯用过程模型^①力求达到软件开发的结构和秩序，其活动和任务都是按照过程的特定指引顺序进行的。但是，对于富于变化的软件世界，这一模型是否合适呢？如果我们抛弃传统过程模型（以及模型所规定的秩序），以一些不够结构化的模型取而代之，是否会使软件工作无法达到协调和一致？

这些问题无法简单回答，但是软件工程师有很大的选择余地。在接下来的章节中，我们将探讨以秩序和一致性作为主要问题的传统过程方法。我们称为“传统”是因为，它规定了一套过程元素——框架活动、软件工程动作、任务、工作产品、质量保证以及每个项目的变更控制机制。每个过程模型还定义了过程流（也称为工作流）——也就是过程元素相互之间关联的方式。

所有的软件过程模型都支持第2章和第3章中描述的通用框架活动，但是每一个模型都对框架活动有不同的侧重，并且定义了不同的过程流以不同的方式执行每一个框架活动（以及软件工程动作和任务）。

关键点 过程模型的作用是减少开发新软件产品时出现的混乱。

网络资源 包括最重要的惯用过程模型的“过程模拟比赛”获奖模型可见于 <http://www.ics.uci.edu/~emilyyo/SimSE/downloads.html>。

4.1.1 瀑布模型

有时候，当从沟通到部署都采用合理的线性工作流方式的时候，可以清楚地理解问题的需求。这种情况通常发生在需要对一个已经存在的系统进行明确定义的适应性调整或是增强的时候（比如政府修改了法规，导致财务软件必须进行相应修改）；也可能发生在极少数新的开发工作上，但是需求必须是准确定义和相对稳定的。

关键点 惯用过程模型定义了一组规定的过程元素和一个可预测的过程工作流。

瀑布模型（waterfall model）又称为经典生命周期（classic life cycle），它提出了一个系统的、顺序的软件开发方法^②，从用户需求规格说明开始，通过策划、建模、构建和部署的过程，最终提供完整的软件支持（图4-1）。

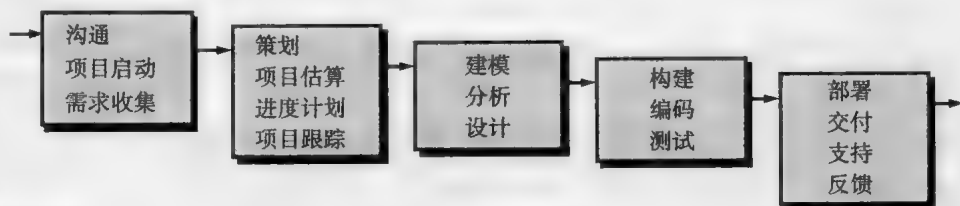


图 4-1 瀑布模型

瀑布模型的一个变体称为 V 模型（V-model）。如图 4-2 所示，V 模型 [Buc99] 描述了质量保证动作同沟通、建模相关动作以及早期构建相关的动作之间的关系。随着软件团队工作沿着 V 模型左侧步骤向下推进，基本问题需求逐步细化，形成了对问题及解决方案的详尽且技术性的描述。一

关键点 V 模型阐明了验证和确认动作如何与早期工程动作相互关联。

① 惯用过程模型有时称为“传统”过程模型。

② 尽管对 Winston Royce [Roy70] 提出的最早的瀑布模型进行了改进，加入了“反馈”循环，但绝大多数软件组织在应用该过程模型时都将其视为严格的线性模型。

一旦编码结束，团队沿着 V 模型右侧的步骤向上推进工作，其本质上是执行了一系列测试（质量保证动作），这些测试验证了团队沿着 V 模型左侧步骤向下推进过程中所生成的每个模型^①。实际上，经典生命周期模型和 V 模型没有本质区别，V 模型提供了一种将验证和确认动作应用于早期软件工程工作中的直观方法。

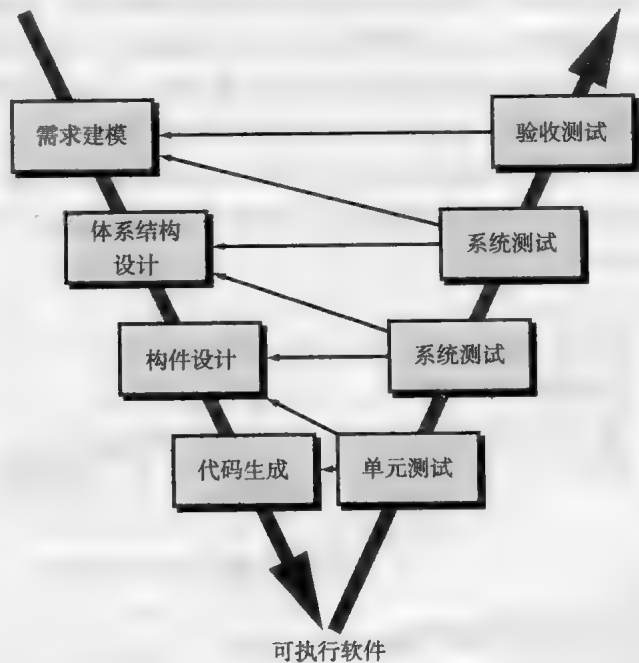


图 4-2 V 模型

瀑布模型是软件工程最早的范例。尽管如此，在过去的 40 多年中，对这一过程模型的批评使它最热情的支持者都开始质疑其有效性 [Han95]。在运用瀑布模型的过程中，人们遇到的问题包括：

1. 实际的项目很少遵守瀑布模型提出的顺序。虽然线性模型可以加入迭代，但是它是用间接的方式实现的，结果是，随着项目组工作的推进，变更可能造成混乱。
2. 客户通常难以清楚地描述所有的需求。而瀑布模型却要求客户明确需求，这就很难适应在许多项目开始阶段必然存在的不确定性。
3. 客户必须要有耐心，因为只有在项目接近尾声的时候，他们才能得到可执行的程序。对于系统中存在的重大缺陷，如果在可执行程序评审之前没有发现，将可能造成惨重损失。

在分析一个实际项目时，Bradac[Bra94]发现，经典生命周期模型的线性特性在某些项目中会导致“阻塞状态”，由于任务之间的依赖性，开发团队的一些成员要等待另一些成员工作完成。事实上，花在等待上的时间可能超过花在生产性工作上的时间。在线性过程的开始和结束，这种阻塞状态更容易发生。

目前，软件工作快速进展，经常面临永不停止的变更流，特性、功能

提问 为什么瀑布模型有时候会失败？

引述 大多数情况下，软件工作遵从骑自行车第一定律：不论你去哪，你都会顶风骑上坡路。

作者不详

① 质量保证动作的详细讨论参见本书第三部分。

和信息内容都会变更，瀑布模型往往并不适合这类工作。尽管如此，在需求已确定的情况下，且工作采用线性的方式完成的时候，瀑布模型是一个很有用的过程模型。

4.1.2 增量过程模型

在许多情况下，初始的软件需求有明确的定义，但是整个开发过程却不宜单纯运用线性模型。同时，可能迫切需要为用户迅速提供一套功能有限的软件产品，然后在后续版本中再进行细化和扩展功能。在这种条件下，需要选用一种以增量的形式生产软件产品的过程模型。

增量模型综合了第3章讨论的线性过程流和并行过程流的特征。如图4-3所示，随着时间的推移，增量模型在每个阶段都运用线性序列。每个线性序列生产出软件的可交付增量 [McD93]。

关键点 增量模型交付一系列称为增量的版本，随着每个版本的交付，逐步为用户提供更多的功能。

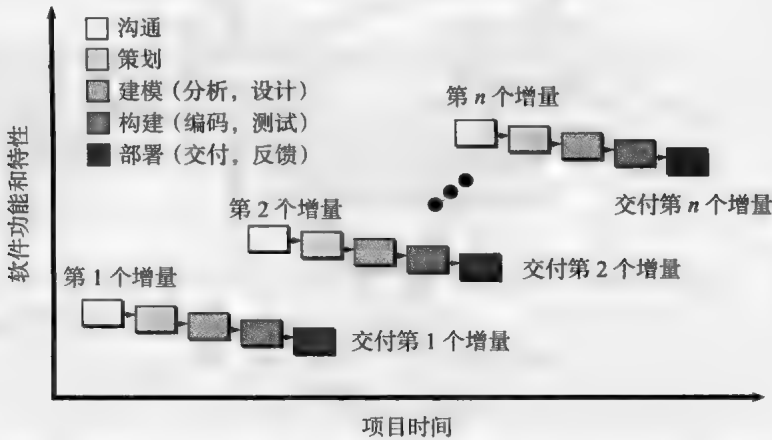


图 4-3 增量模型

例如，采用增量模型开发文字处理软件，在第一个增量中提供基本的文件管理、编辑和文档生成功能；在第二个增量中提供更为复杂的编辑和文档生成功能；在第三个增量中提供拼写和语法检查功能；在第四个增量中提供高级页面排版功能。需要注意的是，任何增量的过程流都可能使用下一节中讨论的原型范型。

运用增量模型的时候，第一个增量往往是核心产品（core product）。也就是满足了基本的需求，但是许多附加的特性（一些是已知的，一些是未知的）没有提供，客户使用该核心产品并进行仔细的评估，然后根据评估结果制定下一个增量计划。这份计划应说明需要对核心产品进行的修改，以便更好地满足客户的要求，也应说明需要增加的特性和功能。每一个增量的交付都会重复这一过程，直到最终产品的产生。

建议 如果你的客户要求你在一个不可能完成的时间提交产品，那么向他建议届时只提交一个或几个增量，此后再提交软件的其他增量。

4.1.3 演化过程模型

软件类似于其他复杂的系统，会随着时间的推移而演化。在开发过程中，商业和产品需求经常发生变化，这将直接导致最终产品难以实现；严格的交付时间使得开发团队不可能圆满完成综合性的软件产品，但是必须交付功能有限的版本以应对竞争或商业压力；虽然能很好地理解核心产品和系统需求，但是产品或系统扩展的细节问题却没有定义。在上述情况和

关键点 演化过程模型中，每个迭代产生软件的一个更完整的版本。

类似情况下，软件开发人员需要一种专门应对不断演变的软件产品的过程模型。

演化模型是迭代的过程模型，这种模型使得软件开发人员能够逐步开发出更完整的软件版本。接下来，将介绍两种常用的演化过程模型。

原型开发。很多时候，客户定义了软件的一些基本任务，但是没有详细定义功能和特性需求。另一种情况下，开发人员可能对算法的效率、操作系统的适用性和人机交互的形式等情况并没有把握。在这些情况和类似情况下，采用**原型开发范型**（prototyping paradigm）是最好的解决办法。

虽然原型可以作为一个独立的过程模型，但是更多的时候是作为一种技术，可以在本章讨论的任何一种过程模型中应用。不论人们以什么方式运用它，当需求很模糊的时候，原型开发模型都能帮助软件开发人员和利益相关者更好地理解究竟需要做什么。

原型开发范型（图 4-4）开始于沟通。软件开发人员和其他利益相关者进行会晤，定义软件的整体目标，明确已知的需求，并大致勾画出以后再进一步定义的东西。然后迅速策划一个原型开发迭代并进行建模（以“快速设计”的方式）。快速设计要集中在那些最终用户能够看到的方面（比如人机接口布局或者输出显示格式）。快速设计产生了一个原型。对原型进行部署，然后由利益相关者进行评估。根据利益相关者的反馈信息，进一步精炼软件的需求。在原型系统不断调整以满足各种利益相关者需求的过程中，采用迭代技术，同时也使开发者逐步清楚用户的需求。

理想状况下，原型系统提供了定义软件需求的一种机制。当需要构建可执行的原型系统时，软件开发人员可以利用已有的程序片段或应用工具快速产生可执行的程序。

如果我们的原型达到了上述目的，那么接下来它有什么用呢？Brooks[Bro95]给出了答案：

在大多数项目中，构建的第一个系统很少是好用的，可能太慢了、太大了、太难用了，或者同时具备上述三点。除了重新开始，没有更好的选择。采用更巧妙的方法来构建一个重新设计的版本，解决上述问题。

原型可以作为第一个系统，也就是 Brooks 推荐我们扔掉的系统。但这可能是一种理想的方式。尽管许多原型系统是临时系统并且会被废弃，但其他一些原型系统将会演化为实际系统。

利益相关者和软件工程师确实都喜欢原型开发范型。客户对实际的系统有了直观的认识，开发者也迅速建立了一些东西。但是，原型开发也存在一些问题，原因如下。

1. 利益相关者看到了软件的工作版本，却未察觉到整个软件是随意搭成的，也未察觉到为了尽快完成软件，开发者没有考虑整体软件质

引述 有时候你很想扔掉一款产品，但是通常情况下，你唯一的选择是考虑如何将它卖给客户。

Frederick P.
Brooks

建议 如果你的客户有一个合理的要求，但是对细节没有思路，那么不妨先开发一个原型。

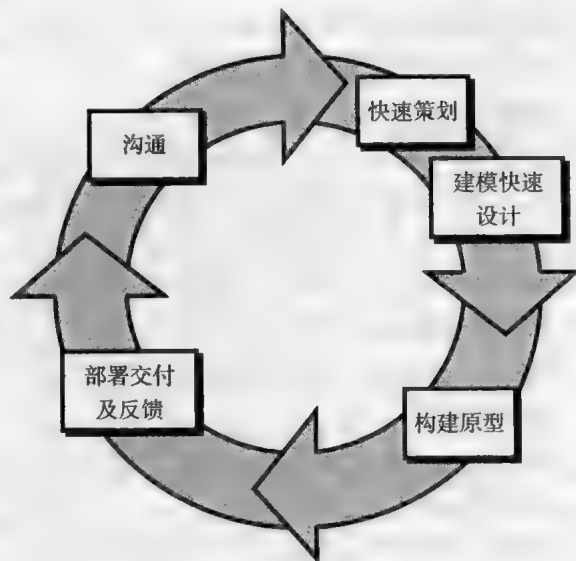


图 4-4 原型开发范型

建议 面对把一个粗糙的原型系统变为工作产品的压力时，如果采取抵制的态度，那么结果往往是产品质量受到损害。

量和长期的可维护性。当开发者告诉客户整个系统需要重建以提高软件质量的时候,利益相关者会不愿意,并且要求对软件稍加修改使其变为一个可运行的产品。在绝大多数的情况下,软件开发管理层会做出妥协。

2. 作为一名软件工程师,为了使一个原型快速运行起来,往往在实现过程中采用折衷的手段。他们经常会使用不合适的操作系统或程序设计语言,仅仅因为当时可用或他们对此较为熟悉。他们也经常会采用一种低效的算法,仅为了证明系统的能力。时间长了,软件开发人员可能会适应这些选择,而忽略了这些选择其实并不合适的理由,结果使并不完美的选择成了系统的组成部分。

尽管问题会发生,但原型开发对于软件工程来说仍是一个有效的范型。关键是要在游戏开始的时候制定规则,也就是说,所有利益相关者必须承认原型是为定义需求服务的。然后丢弃原型(至少是部分丢弃),实际的软件系统是以质量第一为目标而开发的。

SafeHome 选择过程模型 第1部分

[场景] CPI公司软件工程部会议室。该(虚构的)公司专注于开发家用和商用的消费产品。

[人物] Lee Warren, 工程经理; Doug Miller, 软件工程经理; Jamie Lazar、Vinod Raman 和 Ed Robbins, 软件团队成员。

[对话]

Lee: 我简单说一下。正如我们现在所看到的,我已经花了很多时间讨论 SafeHome 产品的产品线。毫无疑问,我们做了很多工作来定义这个东西,我想请各位谈谈你们打算如何做这个产品的软件部分。

Doug: 看起来,我们过去在软件开发方面相当混乱。

Ed: Doug, 我不明白,我们总是能成功开发出产品来。

Doug: 你说的是事实,不过我们的开发工作并不是一帆风顺,并且我们这次要做的项目看起来比以前做的任何项目都要庞大和复杂。

Jamie: 没有你说的那么严重,但是我同意你的看法。我们过去混乱的项目开发方法这次行不通了,特别是这次我们的时间很紧。

Doug (笑): 我希望我们的开发方法更专业一些。我上星期参加了一个培训班,学到了很多关于软件工程的知识。我们现在需要一个过程。

Jamie (皱眉): 我的工作编程,不是文书。

Doug: 在你反对我之前,请先尝试一下。我想说的是……(Doug 开始讲述第3章讲述的过程框架和本章到目前为止讲到的惯用过程模型)

Doug: 所以,似乎线性模型并不适合我们……它假设我们此刻明确了所有的需求,而事实上并不是这样。

Vinod: 同意你的观点。线性模型太IT化了……也许适合于开发一套库存管理系统或者什么,但是不适合我们的 SafeHome 产品。

Doug: 对。

Ed: 原型开发方法听起来不错,正适合我们现在的处境。

Vinod: 有个问题,我担心它不够结构化。

Doug: 别担心。我们还有许多其他选择。我希望在座的各位选出最适合我们小组和我们这个项目的开发范型。

螺旋模型。最早由 Barry Boehm[Boe88] 提出,螺旋模型是一种演进式软件过程模型。它结合了原型的迭代性质和瀑布模型的可控性和系统性特点。它具有快速开发越来越完善的软件版本的潜力。Boehm[Boe01a] 如下这样描述螺旋模型:

螺旋模型是一种风险驱动型的过程模型生成器,对于软件集中的系统,它可以指导多个利益相关者的协同工作。它有两个显著的特点。一是采用循环的方式逐步加深系统定义和实现的深度,同时降低风险。二是确定一系列里程碑作为支撑点,确保利益相关者认可是可行的且可令各方满意的系统解决方案。

螺旋模型将软件开发为一系列演进版本。在早期的迭代中,软件可能是一个理论模型或是原型。在后来的迭代中,会产生一系列逐渐完整的系统版本。

螺旋模型被分割成一系列由软件工程团队定义的框架活动。为了讲解方便,我们使用前文讨论的通用框架活动^①。如图 4-5 所示,每个框架活动代表螺旋上的一个片段。随着演进过程开始,从圆心开始顺时针方向,软件团队执行螺旋上的一圈所表示的活动。在每次演进的时候,都要考虑风险(第 35 章)。每个演进过程还要标记里程碑——沿着螺旋路径达到的工作产品和条件的结合体。

螺旋的第一圈一般开发出产品的规格说明,接下来开发产品的原型系统,并在每次迭代中逐步完善,开发不同的软件版本。螺旋的每圈都会跨过策划区域,此时,需调整项目计划,并根据交付后用户的反馈调整预算和进度。另外,项目经理还会调整完成软件开发需要迭代的次数。

其他过程模型在软件交付后就结束了。螺旋模型则不同,它应用在计算机软件的整个生命周期。因此,螺旋上的第一圈可能表示“概念开发项目”,它起始于螺旋的中心,经过多个迭代^②,直到概念开发的结束。如果这个概念将被开发成为实际的产品,那么该过程将继续沿着螺旋向外伸展,成为“新产品开发项目”。新产品将沿着螺旋通过一系列的迭代不断演进。最后,可以用一圈螺旋表示“产品提高项目”。本质上,当螺旋模型以这种方式进行下去的时候,它将永远保持可操作性,直到软件产品的生命周期结束。过程经常会处于休止状态,但每当有变更时,过程总能够在合适的人口点启动(如产品提高)。

螺旋模型是开发大型系统和软件的很实际的方法。由于软件随着过程的推进而变化,因此在每一个演进层次上,开发者和客户都可以更好地理解 and 应对风险。螺旋模型把原型作为降低风险的机制,更重要的是,开发

关键点 螺旋模型能运用在应用系统开发的整个生命周期,从概念开发到维护。

47

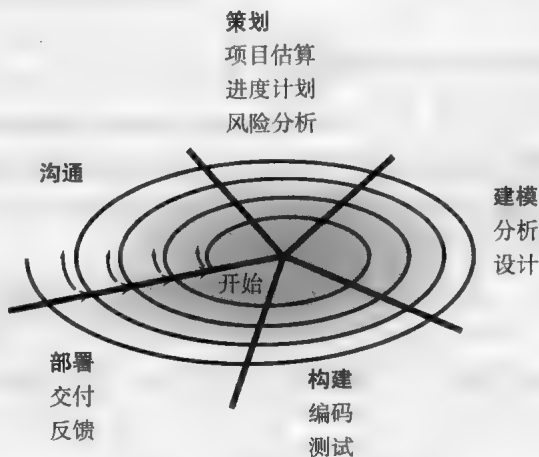


图 4-5 典型的螺旋模型

网络资源 可以在如下网址获得关于螺旋模型的有用信息: www.sei.cmu.edu/publications/documents/00.reports/00sr008.html。

48

建议 如果你的项目要求固定预算开发(通常不是一个好主意),那么螺旋模型会带来问题:每一圈完成的时候,都将重新计划和修改项目开销。

① 这里讨论的螺旋模型有别于 Boehm 提出的模型。原始的螺旋模型可参见 [Boe88]。关于 Boehm 的螺旋模型的更多更新的讨论可参见 [Boe98]。

② 沿轴指向中心的箭头区分开了部署和沟通两个区域,表明沿同一个螺旋路径存在潜在的局部迭代。

人员可以在产品演进的任何阶段使用原型方法。它保留了经典生命周期模型中系统逐步细化的方法，但是把它纳入一种迭代框架之中，这种迭代方式与真实世界更加吻合。螺旋模型要求在项目的所有阶段始终考虑技术风险，如果适当地应用该方法，就能够在风险变为难题之前将其化解。

与其他范型一样，螺旋模型也并不是包治百病的灵丹妙药。很难使客户（特别是以合同的形式）相信演进的方法是可控的。它依赖大量的风险评估专家来保证成功。如果存在较大的风险没有被发现和管理，就肯定会发生问题。

引述 我今天只走这么远，只有明天才能为我指明方向。

Dave Matthews
Band

4.1.4 并发模型

并发开发模型（concurrent development model）有时也叫作并发工程，它允许软件团队表述本章所描述的任何过程模型中的迭代元素和并发元素。例如，螺旋模型定义的建模活动由以下一种或几种软件工动作完成：原型开发、分析和设计。^①

关键点 必须将项目计划看成是活的文档，必须经常对进度进行评估并考虑变更情况，从而对其进行修改。

图 4-6 给出了并发建模方法的一个例子。在某一特定时间，建模活动可能处于图中所示的任何一种状态^②中。其他活动、动作或任务（如沟通或构建）可以用类似的方式表示。所有的软件工程活动同时存在并处于不同的状态。

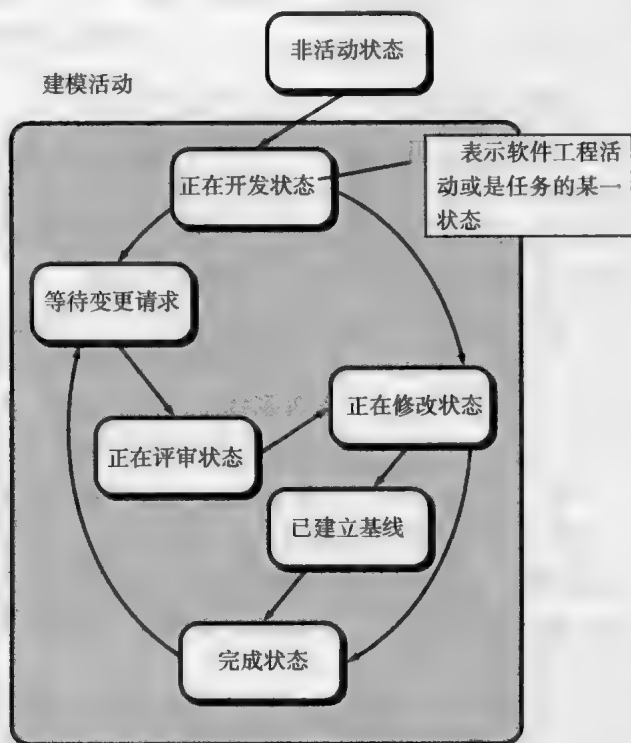


图 4-6 并发过程模型的一个元素

① 需要注意的是，分析和设计都是很复杂的任务，需要进行大量的讨论。本书的第 2 部分将详细讨论这些问题。

② 状态是外部可见的某种行为模式。

例如，在项目的早期，沟通活动（图中并未标明）完成了第一个迭代，停留在等待变更状态。建模活动（初始沟通完成后，一直停留在非活动状态）现在转换到正在开发状态。然而，如果客户要求必须完成需求变更，那么建模活动就会从正在开发状态转换到等待变更状态。

并发建模定义了一系列事件，这些事件将触发软件工程活动、动作或者任务的状态转换。例如，设计的早期阶段（建模活动期间发生的主要软件工程动作）发现了需求模型中的不一致性，于是产生了分析模型修正事件，该事件将触发需求分析动作从完成状态转换到等待变更状态。

并发建模可用于所有类型的软件开发，它能够提供更精确的项目当前状态图。它不是把软件工程活动、动作和任务局限在一个事件的序列，而是定义了一个过程网络。网络上每个活动、动作和任务与其他活动、动作和任务同时存在。过程网络中某一点产生的事件可以触发与每一个活动相关的状态的转换。

建议 并发模型更适合于不同软件工程团队共同开发的产品工程项目。

引述 确保你在组织的每一个过程都有客户，如果没有客户，过程就会变成空中楼阁，失去目标。

V. Daniel Hunt

SafeHome 选择过程模型 第2部分

[场景] CPI 公司软件工程部会议室，该公司生产家用和商用消费类产品。

[人物] Lee Warren，工程经理；Doug Miller，软件工程经理；Vinod 和 Jamie，软件工程团队成员。

[对话]

（Doug 介绍了一些可选的演化模型）

Jamie：我现在有了一些想法。增量模型挺有意义的。我很喜欢螺旋模型，听起来很实用。

Vinod：我赞成。我们交付一个增量产品，听取用户的反馈意见，再重新计划，然后交付另一个增量。这样做也符合产品的特性。我们能够迅速投入市场，然后在每个版本或者说在每个增量中添加功能。

Lee：等等，Doug，你的意思是说我们在螺旋的每一轮都重新生成计划？这样不好，我们需要一个计划，一个进度，然后严格遵守这个计划。

Doug：你的思想太陈旧了，Lee。就像他们说的，我们要现实。我认为，随着我们认识的深入和情况的变化来调整计划更好。这是一种更符合实际的方式。如果制定了不符合实际的计划，这个计划还有什么意义？

Lee（皱眉）：我同意这种看法，可是高管人员不喜欢这种方式，他们喜欢确定的计划。

Doug（笑）：老兄，你应该给他们上一课。

4.1.5 演化过程的最终评述

我们注意到，现代计算机软件总是在持续改变，这些变更通常要求在非常短的期限内实现，并且要充分满足客户—用户的要求。许多情况下，及时投入市场是最重要的管理要求。如果市场时间错过了，软件项目自身可能会变得毫无意义。^①

① 需要注意的是，尽管及时投入市场很重要，但最早进入市场并不保证一定成功。事实上，许多非常成功的软件是第二个或是第三个进入市场的（他们的成功在于吸取了前人的教训）。

演化过程模型就是为了解决上述问题的，但是，作为一类通用的过程模型，它们也有缺点。Nogueira 和他的同事对此概括如下 [Nog00]：

尽管演化软件过程毫无疑问具有一定的优势，但我们还是有一些担忧。首先，由于构建产品需要的周期数目不确定，原型开发（和其他更加复杂的演化过程）给项目计划带来了困难……

其次，演化软件过程没有确定演化的最快速度。如果演化的速度太快，完全没有间歇时间，项目肯定会陷入混乱；反之，如果演化速度太慢，则会影响生产率……

再次，演化软件过程应该侧重于灵活性和可延展性，而不是高质量。这种说法听起来很惊人。

确实，一个强调灵活性、可扩展性和开发速度而不是高质量的软件过程听起来令人震惊。可是，很多广为人们尊重的软件工程专家都这样建议（例如 [You95]、[Bac97]）。

演化模型的初衷是采用迭代或者增量的方式开发高质量软件^①。可是，用演化模型也可以做到强调灵活性、可延展性和开发速度。软件团队及其经理所面临的挑战就是在这些严格的项目、产品参数与客户（软件质量的最终仲裁者）满意度之间找到一个合理的平衡点。

提问 演化过程模型具有哪些不为人知的弱点？

4.2 专用过程模型

专用过程模型具有前面章节中提到的传统过程模型的一些特点，但是，专用过程模型往往应用面较窄且较专一，只适用于某些特定的软件工程方法。^②

4.2.1 基于构件的开发

商业现货（Commercial Off-The-Shelf, COTS）软件构件由厂家作为产品供应，通过良好定义的接口提供特定的功能，这些构件能够集成到正在构建的软件中。基于构件的开发模型（component-based development model）具有许多螺旋模型的特点。它本质上是演化模型 [Nie92]，需要以迭代方式构建软件。不同之处在于，基于构件的开发模型采用预先打包的软件构件来开发应用系统。

网络资源 基于构件开发的相关信息可以参见 www.cbd-hq.com。

建模和构建活动开始于识别可选构件。这些构件有些设计成传统的软件模块，有些设计成面向对象的类或类包^③。若不考虑构件的开发技术，则基于构件开发模型由以下步骤组成（采用演化方法）：

1. 对于该问题的应用领域研究和评估可用的基于构件的产品。
2. 考虑构件集成的问题。
3. 设计软件架构以容纳这些构件。
4. 将构件集成到架构中。
5. 进行充分的测试以保证功能正常。

① 在这里，软件质量的含义非常广泛，不仅包括客户满意度，还包括本书第二部分讲的各种技术指标。

② 在某些情况下，这些专用过程模型也许应该更确切地称为技术的集合或“方法论”，是为了实现某一特定的软件开发目标而制定的。但它们确实也提出了一种过程。

③ 附录 2 讨论了面向对象概念，该概念的使用贯穿了本书第二部分。在这里，类封装了一组数据以及处理数据的过程。类包是一组共同产生某种结果的相关类的集合。

基于构件的开发模型能够使软件复用,从而为软件工程师带来极大收益。如果构件复用已经成为你所在的软件工程团队文化的一部分,那么将会缩短开发周期并减少项目开发费用。基于构件的软件开发将在第14章进行详细讨论。

4.2.2 形式化方法模型

形式化方法模型(formal methods model)的主要活动是生成计算机软件形式化的数学规格说明。形式化方法使软件开发人员可以应用严格的数学符号来说明、开发和验证基于计算机的系统。这种方法的一个变形是净室软件工程(cleanroom software engineering) [Mil87, Dye92],这一软件工程方法目前已应用于一些软件开发机构。

形式化方法(附录3)提供了一种机制,使得在软件开发中可以避免一些问题,而这些问题在使用其他软件工程模型时是难以解决的。使用形式化方法时,歧义性问题、不完整问题、不一致问题等都能够更容易地被发现和改正——不是依靠特定的评审,而是应用数学分析的方法。在设计阶段,形式化方法是程序验证的基础,使软件开发人员能够发现和改正一些常常被忽略的问题。

虽然形式化方法不是一种主流的方法,但它的意义在于可以提供无缺陷的软件。尽管如此,人们还是对在商业环境中应用形式化方法有怀疑,这表现在:

- 目前,形式化模型开发非常耗时,成本也很高。
- 只有极少数程序员具有应用形式化方法的背景,因此需要大量的培训。
- 对于技术水平不高的客户,很难用这种模型进行沟通。

尽管有这些疑虑,但软件开发中还是有很多形式化方法的追随者,比如有人用其开发那些高度关注安全的软件(如飞行器和医疗设施),或者开发那些一旦出错就将导致重大经济损失的软件。

提问 既然形式化方法能够保证软件的正确性,为什么没有被广泛应用呢?

4.2.3 面向方面的软件开发

不论选择什么软件过程,复杂软件都无一例外地实现了一套局部化的特性、功能和信息内容。这些局部的软件特性被做成构件(例如面向对象的类),然后在系统架构中使用。随着现代计算机系统变得更加复杂,某些关注点——客户需要的属性或者技术兴趣点——已经体现在整个架构设计中。有些关注点是系统的高层属性(例如安全性、容错能力),另一些关注点影响了系统的功能(例如商业规则的应用),还有一些关注点是系统性的(例如任务同步或内存管理)。

如果某个关注点涉及系统多个方面的功能、特性和信息,那么这些关注点通常称为横切关注点(crosscutting concern)。方面的需求(aspectual requirement)定义那些对整个软件体系结构产生影响的横切关注点。面向方面的软件开发(Aspect-Oriented Software Development, AOSD)通常称为面向方面编程(Aspect-Oriented Programming, AOP)或者面向方面构件工程(Aspect-Oriented Component Engineering, AOCE) [Gru02],它是相对较新的一种软件工程模型,为定义、说明、设计和构建方面(aspect)提供过程和方法——“是对横切关注点进行局部表示的一种机制,超越了子程序和继承方法” [Elr01]。

网络资源 关于AOP的资源和信息可以参见aosd.net。

关键点 AOSD定义了“方面”,表示用户跨越多个系统功能、特性和信息的关注点。

与众不同的面向方面的过程还不成熟。尽管如此,这种过程模型看似具备了演化模型和并发过程模型的共同特点。演化模型适合定义和构建方面;而并发开发的并行特点很重要,因为方面是独立于局部的软件构件开发的,并且对这些构件的开发有直接影响。因此,在构建方面和构件的过程活动之间建立起异步的通信非常重要。

关于面向方面的软件开发最好查阅相关专著中的详细讨论。感兴趣的读者可以参看[Ras11]、[Saf08]、[Cla05]、[Fil05]、[Jac04]和[Gra03]。

软件工具 过程管理

[目标] 辅助定义、执行和管理传统过程模型。

[机制] 过程管理工具帮助软件组织或团队定义完整的软件过程模型(框架活动、动作、任务、质量保证检查点、里程碑和工作产品)。而且,该工具为软件工程师的技术工作提供路线图,为经理们跟踪和控制软件过程提供模板。

[代表性工具]^①

- GDPA。一个研究性的过程定义工具包,该工具包由德国的 Bremen 大学(www.informatik.uni-bremen.de/uniform/gdpa/home.htm)开发,它提供了大量的过程

建模和管理功能。

- ALM Studio。由 Kovair 公司(<http://www.kovair.com/>)开发的一套工具包,用于过程定义、需求管理、问题解决、项目策划和跟踪。
- ProVision BPMx。由 OpenText(<http://bps.opentext.com>)开发,它提供了很多代表性工具,可以辅助过程定义和工作流自动化。

以下网站提供了很多与软件过程相关的各种很有价值的工具:www.computer.org/portal/web/swbok/html/ch10。

4.3 统一过程

Ivar Jacobson、Grady Booch 和 James Rumbaugh[Jac99]在他们关于统一过程(unified process)的影响深远的著作中,讨论了关于有必要建立一种“用例驱动,以架构为核心,迭代并且增量”的软件过程的问题,并阐述如下:

当前,软件朝着更大、更复杂的系统发展。部分原因在于计算机的计算能力逐年递增,使得人们对其的期望值增大。另一方面,还是受 Internet 应用膨胀的影响,促使各类信息交换……我们从软件版本的提升中学会了如何改进产品,使我们对更复杂软件的胃口越来越大。同时希望软件更好地满足我们的需要,结果导致软件更加复杂。简而言之,我们想要的越来越多。

在某种程度上,统一过程尝试着从传统的软件过程中挖掘最好的特征和性质,但是以敏捷软件开发(第5章)中许多最好的原则来实现。统一过程认识到与客户沟通以及从用户的角度描述系统(即用例)^②并保持该描述的一致性的的重要性。它强调软件体系结构的重要

① 这里提到的工具只是此类工具的例子,并不代表本书支持采用这些工具。大多数情况下,工具的名字由各自的开发者注册为商标。

② 用例(use case)(第5章)是一种文字描述或模板,从用户的角度描述系统功能和特性。用例由用户来写,并作为创建更为复杂的分析模型的基础。

作用，并“帮助架构师专注于正确的目标，例如可理解性、对未来变更的可适应性以及复用” [Jac99]。它建立了迭代的、增量的过程流，提供了演进的特性，这对现代软件开发非常重要。

4.3.1 统一过程的简史

20 世纪 90 年代早期，James Rumbaugh [Rum91]、Grady Booch [Boo94] 和 Ivar Jacobson [Jac92] 开始研究“统一方法”，他们的目标是结合各自面向对象分析和设计方法中最好的特点，并吸收其他面向对象模型专家提出的其他特点（例如 [Wir90]）。他们的成果就是 UML——统一建模语言（unified modeling language），这种语言包含了大量用于面向对象系统建模和开发的符号。到了 1997 年，UML 已经变成了面向对象软件开发的行业标准。

UML 作为需求模型和设计模型的表示方式，其应用贯穿本书第二部分。附录 1 针对不熟悉 UML 基本概念和建模规则的人给出了介绍性的指导。有关 UML 的全面介绍可以参考有关 UML 的材料，附录 1 中列出了相关书籍。

4.3.2 统一过程的阶段^①

在第 3 章，我们讨论了 5 种通用的框架活动，并认为它们可以用来描述任何软件过程模型。统一过程也不例外。图 4-7 描述了统一过程（Unified Process, UP）的“阶段”，并将他们与第 1 章及本章前面部分讨论的通用活动进行了对照。

UP 的起始阶段（inception phase）包括客户沟通和策划活动。通过与利益相关者协作定义软件的业务需求，提出系统大致的架构，并制定开发计划以保证项目开发具有迭代和增量的特性。该阶段识别基本的业务需求，并初步用用例（第 8 章）描述每一类用户所需要的主要特性和功能。此时的体系结构仅是主要子系统及其功能、特性的试探性概括。随后，体系结构将被细化和扩充成为一组模型，以描述系统的不同视图。策划阶段将识别各种资源，评估主要风险，制定进度计划，并为其在软件增量开发的各个阶段中的应用建立基础。

细化阶段（Elaboration Phase）包括沟通和通用过程模型的建模活动（图 4-7）。细化阶段扩展了初始阶段定义的用例，并扩展了体系结构以包括软件的五种视图——用例模型、需求模型、设计模型、实现模型和部署模型。在某些情况下，细化阶段建立了一个“可执行的体系结构基线” [Arl02]，这是建立可执行系统的“第一步”（first cut）^②。体系结构基线证明了体系结构的可实现性，但没有提供系统使用时所需的所有功能和特性。另

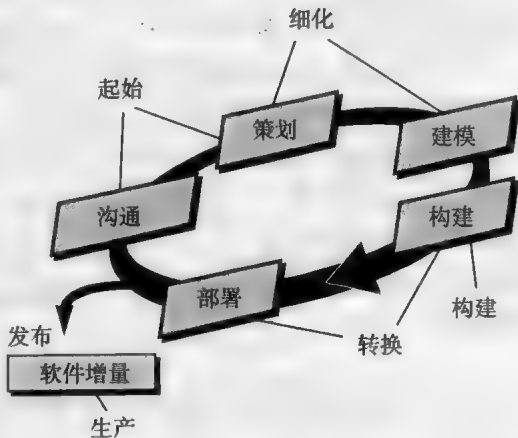


图 4-7 统一过程

关键点 UP 阶段的目的与本书中定义的通用框架活动的目的是类似的。

① 统一过程有时也用 Rational 公司（后来被 IBM 收购）的名字命名，称为 Rational 统一过程（Rational Unified Process, RUP），Rational 公司是早期开发和细化该统一过程的主要投资方，并建立了支持该过程的完整环境（工具及技术）。

② 需要指出的是，体系结构基线不是原型系统，因为它并不会被抛弃，而是在下一个 UP 阶段进一步充实。

外，在细化的最终阶段将评审项目计划以确保项目的范围、风险和交付日期的合理性。该阶段通常要对项目计划进行修订。

UP 的构建阶段（construction phase）与通用软件过程中的构建活动相同。构建阶段采用体系结构模型作为输入，开发或是获取软件构件，使得最终用户能够操作用例。为达到上述目的，要对在细化阶段开始的需求模型和设计模型加以完善，以反映出软件增量的最终版本。软件增量（例如发布的版本）所要求的必须具备的特性和功能在源代码中实现。随着构件的实现，对每一个构件设计并实施单元测试^①。另外，还实施了其他集成活动（构件组装和集成测试）。用例用于导出一组验收测试，以便在下一个 UP 阶段开始前执行。

网络资源 敏捷开发中统一过程的有趣讨论参见 www.ambysoft.com/unifiedprocess/agileUP.html。

UP 的转换阶段（transition phase）包括通用构建活动的后期阶段以及通用部署（交付和反馈）活动的第一部分。软件被提交给最终用户进行 Beta 测试，用户反馈报告缺陷及必要的变更。另外，软件开发团队创建系统发布所必需的支持信息（如用户手册、问题解决指南及安装步骤）。在转换阶段结束时，软件增量成为可用的发布版本。

UP 的生产阶段（production phase）与通用过程的部署活动一致。在该阶段，对持续使用的软件进行监控，提供运行环境（基础设施）的支持，提交并评估缺陷报告和变更请求。

有可能在构建、转换和生产阶段的同时，下一个软件增量的工作已经开始。这就意味着五个 UP 阶段并不是顺序进行，而是阶段性地并发进行。

软件工程的工作流分布在所有 UP 阶段。在 UP 中，工作流类似于任务集（参见第 3 章的描述）。也就是说，工作流识别了完成一个重要的软件工程活动的必要任务，以及在完成任务之后所产生的工作产品。需要注意的是，并不是工作流所识别的每一个任务都在所有的项目中应用。软件开发团队应根据各自的需要适当调整过程（动作、任务、子任务及工作产品）。

58

4.4 个人过程模型和团队过程模型

好的软件过程需要贴近于过程的执行人员。如果一个软件过程模型是为社团或是软件组织制定的，那么，只有对其进行充分改进，并使其真正满足实施软件工程项目的要求，该模型才能有效。理想情况下，软件开发人员会建立最适合他们的过程，并同时能够与开发团队及整个组织的要求相吻合。换句话说，开发团队将制定其自己的过程，同时满足个人的小范围的要求和企业的大范围要求。Watts Humphrey([Hum97] 和 [Hum00]) 认为，有可能建立“个人软件过程”和“团队软件过程”。虽然二者都需要艰苦努力、培训和协调，但都是可以做到的。^②

引述 成功者不过是养成了成功人士做事的习惯。
Dexter Yager

4.4.1 个人软件过程

每个开发人员都采用某种过程来开发计算机软件。这种过程也许是随机的，也许是特定

① 有关软件测试（包括单元测试）的深入讨论参见第 22 ~ 26 章。

② 需要指出的是，敏捷方法的支持者（第 5 章）同样认为过程应贴近团队。他们只不过提出了达到同样目的的另外一种解决方法。

的,可能每天都会改变,可能不够高效、不够有效甚至不成功,但不管怎样,过程都是存在的。Watts Humphrey[Hum05]建议为了改变无效的个人过程,开发人员必须经过四个阶段,每个阶段都需要培训和认真操作。个人软件过程(Personal Software Process, PSP)强调对产品以及产品质量的个人测量。并且, PSP让第一线工作人员负责项目计划的制定(如估算和进度安排),并授权给他们来控制所有开发的软件产品的质量。PSP过程模型定义了以下五个框架工作活动。

策划。它将需求活动分离出来,并估算项目的规模和所需资源,并且进行缺陷评估(估计此项工作的缺陷数目)。所有的度量都用工作表或是模板记录。最后,识别开发任务并建立项目进度计划。

高层设计。建立每个构件的外部规格说明并完成构件设计。如果有不确定的需求,则构建原型系统。所有问题都要记录和跟踪。

高层设计评审。使用正式的验证方法(附录3)来发现设计中的错误。对所有的重要任务和工作结果都进行度量。

开发。细化和评审构件级设计。完成编码,对代码进行评审,并进行编译和测试。对所有的重要任务和工作结果都进行度量。

后验。根据收集到的度量和测量结果(需要进行大量数据的统计分析)确定过程的有效性。度量和测量结果为提高过程的有效性提供指导。

PSP强调尽早发现错误,并且分析易犯错误的种类,后者和前者同样重要。这是通过对软件开发人员提交的所有产品进行严格评估实现的。

PSP代表的是一种严格有序的、基于度量的软件工程方法,这可能会对许多第一线工作人员产生文化冲击。但是,如果将PSP恰当地介绍给软件工程师[Hum96],那么软件工程的生產率和软件质量将大幅度提高[Fer97]。然而PSP没有被工业界广泛地加以采纳。遗憾的是,其原因更主要的在于人的本性和软件开发组织的惯性,而非PSP方法本身的好坏。PSP是对能力的极大挑战,并且需要得到一定程度的承诺(包括第一线工作人员及其经理),这种支持通常很难得到。PSP的培训相对时间较长,价格较高。由于文化的关系,对很多软件人员来说都难以达到所需的度量水平。

PSP能否用作个人的有效软件过程呢?答案尚不明确。但是,即使PSP没有得到完全的采纳,但它所引进的许多个人过程改进的理念也值得学习。

4.4.2 团队软件过程

考虑到很多行业级软件项目都由项目团队开发,Watts Humphrey吸取了引入PSP的经验教训,并提出了团队软件过程(Team Software Process, TSP)。TSP的目标是建立一个能够“自我管理”的项目团队,团队能自我组织进行高质量的软件开发。Humphrey[Hum98]为TSP定义了以下目标:

- 建立自我管理团队来策划和跟踪其工作、确定目标、建立团队自己的过程和计划。团队既可以是纯粹的软件开发队伍,也可以是集成的产品队伍(Integrated Product Team, IPT),可以由3~20名工程师组成。
- 指示管理人员如何指导和激励其团队,并保持团队的最佳表现。

网络资源 有关PSP的大量资源可见于<http://www.sei.cmu.edu/tsp/tools/academic/>。

提问 PSP中用到了哪些框架活动?

关键点 PSP强调对所犯的错误类型进行记录和分析,以便制定消除错误的策略。

网络资源 关于采用PSP和TSP构建优秀团队的信息可参考www.sei.cmu.edu/tsp/。

- 使 CMM^①第 5 级的行为常规化，并依此约束员工，这样可加速软件过程改进。
- 为高成熟度的软件组织提供改进指导。
- 协助大学教授行业级团队技能。

一个自我管理的团队对其整体目标有一致的理解。它定义了每个团队成员的角色和责任；跟踪量化的项目数据（包括生产率和质量）；确定适合该项目的团队过程和执行该过程的具体策略；定义适合团队软件工程工作的本地标准；持续评估风险并采取风险规避措施；跟踪、管理和报告项目状态。

建议 为了组建有自我管理能力的团队，必须能够做到内部相互配合，并与外部建立良好沟通。

TSP 定义了以下框架活动：项目启动、高层设计、实现、集成和测试以及后验。正如在 PSP 中与其对应的活动（注意这里用的术语是不一样的），这些活动使整个团队按规范的方式计划、设计和构建软件，同时量化地评测软件过程和产品。后验用于确定过程改进的步骤。

TSP 使用大量的脚本、表格和标准等来指导其团队成员的工作。脚本（script）定义了特定的过程活动（如项目启动、设计、实现、集成和系统测试、后验），以及作为团队过程一部分的其他更详细的工作职能（如开发计划的制定、需求开发、软件配置管理及单元测试）。

关键点 TSP 脚本定义了团队过程的组成部分，以及过程中的活动。

TSP 认为，最好的软件团队需要具有自我管理的能力^②。团队成员制定项目目标，调整过程以满足需要，控制进度计划，并通过度量及对所收集的度量值的分析，不断改进团队的软件工程方法。

与 PSP 类似，TSP 也是一个严格的软件工程过程，在提高生产率和质量方面可以取得明显的和量化的效果。开发团队必须对过程做出全面的承诺，并且经过严格的训练以保证方法得以很好地应用。

4.5 过程技术

前面章节中讨论的一些过程模型必须加以调整才能应用于特定的软件项目团队。为了利于模型调整，人们开发了过程技术工具（process technology tool）以帮助软件开发组织分析现有过程、组织工作任务、控制并监控过程进度和管理技术质量。

过程技术工具帮助软件开发组织对第 3 章中讨论的过程框架、任务集和普适性活动建造自动化的模型。模型通常用网络图表示，可以通过分析定义典型的工作流，并识别有可能减少开发时间或费用的其他可选择的过程结构。

一旦创建了可接受的过程，其他过程技术工具可用来分配、监控甚至控制过程模型中定义的所有软件工程活动、动作和任务。每个项目组成员都可以利用这种工具建立需要完成的工作任务检查单、工作产品检查单和需要执行的质量保证活动检查单。过程技术工具也可以和其他适用于特定工作任务的软件工程工具配合使用。

① 能力成熟度模型（CMM）是一种衡量软件过程效率的技术，将在第 37 章进行讨论。

② 第 5 章讨论了敏捷模型中的关键因素“自我管理”团队的重要性。

软件工具 过程建模工具

[目标] 软件组织必须首先理解软件过程，然后才能对其进行改进。过程建模工具（也称为过程技术工具或是过程管理工具）用来表示过程的关键环节，以便更好地理解过程。这些工具也可以提供对于过程描述的链接，以协助过程的参与人员了解并掌握所需完成的操作及工作任务。过程建模工具还提供了与其他工具的链接，以支持所定义的过程活动。

[机制] 这类工具辅助开发团队定义一个特定过程模型的组成部分（如动作、任务、工作产品、质量保证点等），为每个过程元素的描述和内容定义提供详细的指导，

并管理过程执行。在某些情况下，过程技术工具包括标准的项目管理任务，如估算、进度计划、跟踪和控制。

[代表性工具]^①

- Igrafx Process Tools。对软件过程进行映射、度量和建模的一组工具（<http://www.igrafx.com/>）。
- Adeptia BPM Server。用来管理、增强自动化、优化业务过程的工具（www.adeptia.com）。
- ALM Studio Suite。一个重点关注对沟通 and 建模活动进行管理的工具集（<http://www.kovair.com>）。

4.6 产品和过程

如果过程很薄弱，则最终产品必将受到影响。但是对过程的过分依赖也是很危险的。Margaret Davis[Dav95a] 在多年前写的一篇简短的文章里对产品和过程的双重性进行了以下评述：

大约每十年或五年，软件界都会对“问题”重新定义，其重点由产品问题转向了过程问题。因此，我们逐步采纳了结构化程序设计语言（产品）、结构化分析方法（过程）和数据封装（产品），到现在重点是卡内基·梅隆大学软件工程研究所提出的能力成熟度模型（过程）（随后逐步采纳面向对象方法和敏捷软件开发）。

钟摆的自然趋势是停留在两个极端的中点，与之类似，软件界的关注点也不断地摆动，当上一次摆动失败时，就会有新的力量加入，促使它摆向另一个方向。这些摆动是非常有害的，因为它们可能从根本上改变了工作内容及工作方法，使软件工程实践人员陷入混乱。而且这些摆动并没有解决问题，只是把产品和过程分裂开来而不是作为辩证统一的一体，因此注定要失败。

这种二象性在科学界早有先例，当某一个理论不能对观测到的相互矛盾的结果做出合理解释时，就会出现二象性理论。由 Louis de Broglie 于 20 世纪 20 年代提出的光的波粒二象性就是一个很好的例子。我相信，我们对软件组成部分和开发过程的观测证明了软件具有过程和产品的二象性。如果仅仅将软件看作一个过程或是一个产品，那就永远都不能正确地理解软件，包括其背景、应用、意义和价值。

所有的人类活动都可以看成一个过程，我们每一个人都从这些活动中获得对自我价值的认识，这些活动所产生的结果可以被许多人反复地在不同的情况下使用。也就是说，我们是

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

从我们自己或是他人对我们产品的复用中得到满足的。

因此，将复用目标融入软件开发，这不仅潜在地增加了软件专业人员从工作中获得的满足感，也增加了接受“产品和过程二象性”这一观点的紧迫性。对于一个可复用的部件，如果仅仅从产品或是仅仅从过程的角度考虑，都不利于软件开发，这种片面的观点或者影响了人们对产品的应用环境 and 应用方法的认识，或者忽略了该产品还可以作为其他开发活动的输入这一事实。因此，片面地强调某一方面的观点会极大地降低软件复用的可能性，也会大大减少工作的成就感。

正如从最终产品获得满足一样，人们在创造性的过程中得到了同样的（甚至更大的）成就感。艺术家不仅仅对装裱好的画卷感到高兴，更在每一笔绘画的过程中享受乐趣；作家不仅欣赏出版的书籍，更为每一个苦思得到的比喻而欣喜。一个具有创造性的专业软件人员也应该从过程中获得满足，其程度不亚于最终的产品。产品和过程的二象性已经成为保留推动软件工程不断进步的创造性人才的一个重要因素。

63

4.7 小结

传统软件过程模型已经使用了多年，力图给软件开发带来秩序和结构。每一个模型都建议了一种不同的过程流，但所有模型都实现同样的一组通用框架活动：沟通、策划、建模、构建和部署。

像瀑布模型和 V 模型这类顺序过程模型是最经典的软件工程模型，顺序过程模型建议采用线性过程流，这在软件世界里通常与当代的软件开发现实情况不符（例如持续的变更、演化的系统、紧迫的开发时间）。但线性过程模型确实适用于需求定义清楚且稳定的软件开发。

增量过程模型采用迭代的方式工作，能够快速生成一个软件版本。演化过程模型认识到大多数软件项目的迭代、递增特性，其设计的目的是为了适应变更。演化模型，例如原型开发及螺旋模型，会快速地产生增量的工作产品（或是软件的工作版本）。这些模型可以应用于所有的软件工程活动——从概念开发到长期的软件维护。

并发过程模型为软件团队提供了过程模型中的重叠和并发元素的描述方法。专用模型主要包括基于构件的模型，强调软件构件的重用和组装；形式化方法模型提倡采用数学方法进行软件开发与验证；面向方面的模型的目的是解决跨越整个软件体系结构的横切关注问题。统一过程模型是一种“用例驱动、以体系结构为核心、迭代及增量”的软件过程框架，由 UML 方法和工具支持。

软件过程的个人模型和团队模型都强调成功软件过程的关键因素：测量、策划和自我管理。

习题与思考题

- 4.1 详细描述三个适于采用瀑布模型的软件项目。
- 4.2 详细描述三个适于采用原型模型的软件项目。
- 4.3 如果将原型变成一个可发布的系统或者产品，应该如何调整过程？
- 4.4 详细描述三个适于采用增量模型的软件项目。
- 4.5 当沿着螺旋过程流发展的时候，你对正在开发或者维护的软件的看法是什么？
- 4.6 可以合用几种过程模型吗？如果可以，举例说明。

64

- 4.7 并发过程模型定义了一套“状态”，用你自己的话描述一下这些状态表示什么，并指出他们在并发过程模型中的作用。
- 4.8 开发质量“足够好”的软件，其优点和缺点是什么？也就是说，当我们追求开发速度胜过产品质量的时候，会产生什么后果？
- 4.9 详细描述三个适于采用基于构件模型的软件项目。
- 4.10 我们可以证明一个软件构件甚至整个程序的正确性，可是为什么并不是每个人都这样做？
- 4.11 统一过程和 UML 是同一概念吗？解释你的答案。

扩展阅读与信息资源

第2章的扩展阅读部分提到的大部分资料都详细地介绍了传统过程模型。

Cynkovic 和 Larsson (《Building Reliable Component-Based Systems》，Addison-Wesley, 2002) 以及 Heineman 和 Council (《Component-Based Software Engineering》，Addison-Wesley, 2001) 描述了实现基于构件系统的过程需求。Jacobson 和 Ng (《Aspect-Oriented Software Development with Use Cases》，Addison-Wesley, 2005) 以及 Filman 和他的同事 (《Aspect-Oriented Software Development》，Addison-Wesley, 2004) 讨论了面向方面过程的独特性质。Monin 和 Hinchey (《Understanding Formal Methods》，Springer, 2003) 提供了有价值的介绍，Baco 和他的同事 (《Formal Methods》，Springer, 2009) 讨论了目前的最新水平和新方向。

Kenett 和 Baker (《Software Process Quality : Management and Control》，Marcel Dekker, 1999) 以及 Chrissis、Konrad 和 Shrum (《CMMI for Development : Guidelines for Process Integration and Product Improvement》，3rd ed., Addison-Wesley, 2011) 考虑了高质量的管理和过程设计是如何相互影响的。

除了 Jacobson、Rumbaugh 和 Booch 的有关统一过程的书籍 [Jac99] 以外，Shuja 和 Krebs (《IBM Rational Unified Process Reference and Certification Guide》，IRM Press, 2008)、Arlow 和 Neustadt (《UML 2 and the Unified Process》，Addison-Wesley, 2005)、Kroll 和 Kruchten (《The Rational Unified Process Made Easy》，Addison-Wesley, 2003) 以及 Farve (《UML and the Unified Process》，IRM Press, 2003) 等人的书籍提供了很好的补充信息。Gibbs (《Project Management with the IBM Rational Unified Process》，IBM Press, 2006) 讨论了统一过程中的项目管理问题。Dennis、Wixom 和 Tegarden (《System Analysis and Design with UML》，4th ed., Wiley, 2012) 解决涉及 UP 的编程和业务过程建模问题。

网上有大量关于软件过程模型的信息源，与软件过程有关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

敏捷开发

要点浏览

概念：敏捷软件工程是哲学理念和一系列开发指南的综合。这种哲学理念推崇：让客户满意且尽早的增量发布；小而高度自主的项目团队；非正式的方法；最小化软件工作产品以及整体精简开发。开发的指导方针强调超越分析和设计（尽管并不排斥这类活动）的发布，以及开发人员和客户之间主动和持续的沟通。

人员：软件工程师和其他项目利益相关者（经理、客户、最终用户）共同组成敏捷开发团队，这个团队是自我组织的并掌握着自己的命运。敏捷团队鼓励所有参与人员之间的交流与合作。

重要性：孕育着基于计算机的系统和软件产品的现代商业环境正以飞快的节奏不断变化着。敏捷软件工程提出了针对特定类型软件和软件项目的不同于传统软

件工程的合理方案。事实证明，这一方法可以快速交付成功的系统。

步骤：对敏捷开发的恰当称呼应当是“软件工程精简版”，它保留了基本的框架活动：客户沟通、策划、建模、构建和部署，但将其缩减到一个推动项目组朝着构建和交付发展的最小任务集（有人认为这种方法是牺牲问题分析和方案设计为代价而实现的）。

工作产品：客户和软件工程师有着共同的观点——唯一真正重要的工作产品是在合适的时间交付给客户的可运行软件增量。

质量保证措施：如果敏捷团队认为过程可行，并且开发出的可交付软件增量能使客户满意，则软件质量就是没有问题的。

2001 年，Kent Beck 和其他 16 位知名软件开发者、软件工程作家以及软件咨询师 [Bec01]（被称为敏捷联盟）共同签署了“敏捷软件开发宣言”。该宣言声明：

我们正在通过亲身实践以及帮助他人实践的方式来揭示更好的软件开发之路，通过这项工作，我们认识到：

- 个人和他们之间的交流胜过了开发过程和工具
- 可运行的软件胜过了宽泛的文档
- 客户合作胜过了合同谈判
- 对变更的良好响应胜过了按部就班地遵循计划

也就是说，虽然上述右边的各项很有价值，但我们认为左边的各项具有更大的价值。

一份宣言通常和一场即将发生的破旧立新的政治运动（期望好转）相关联。从某些方面来讲，敏捷开发确实是这样一场运动。

关键概念

验收测试
敏捷联盟
敏捷过程
敏捷统一过程
敏捷
敏捷原则
变更成本
动态系统开发方法 (DSDM)
极限编程 (XP)
工业 XP
结对编程
敏捷开发战略
项目速度

虽然多年来大家一直都在使用着指导敏捷开发的基本思想，但真正将它们凝聚到一场“运动”中还不到二十年。从本质上讲，敏捷方法^①是为了克服传统软件工程中认识和实践的弱点而形成的。敏捷开发可以带来多方面的好处，但它并不适用于所有的项目、所有的产品、所有的人和所有的情况。它并不完全对立于传统软件工程实践，也不能作为超越一切的哲学理念而用于所有软件工作。

在现代经济生活中，通常很难甚至无法预测一个基于计算机的系统（如移动 App）如何随时间推移而演化。市场情况变化迅速，最终用户需求不断变更，新的竞争威胁毫无征兆地出现。在很多情况下，在项目开始之前，我们无法充分定义需求。因此，我们必须足够敏捷地去响应不断变化、无法确定的商业环境。

不确定性意味着变更，而变更意味着付出昂贵的成本，特别是在其失去控制或疏于管理的情况下，为这种变更而付出的成本费用是昂贵的。而敏捷方法最具强制性的特点之一就是它能够通过软件过程来降低由变更所引起的代价。

难道说认识到现实的挑战，我们就完全抛弃那些有价值的软件工程原理、概念、方法和工具吗？绝对不是。和其他所有工程学科一样，软件工程也在持续发展着，我们可以通过改进软件工程本身来适应敏捷带来的挑战。

Alistair Cockburn[Coc02] 在他那本发人深省的敏捷软件开发著作中，论证了本书第 4 章介绍的惯例过程模型中存在的主要缺陷：忘记了开发计算机软件的人员的弱点。软件工程师不是机器人，许多软件工程师在工作方式上有很大差别，在技能水平、主动性、服从性、一致性和责任心方面也有巨大差异。有些人可以通过书面方式很好地沟通，而有些人则不行。Cockburn 论证说：过程模型可以“利用纪律或者宽容来处理人的共同弱点”，因而大多数惯例过程模型选择了纪律。他还指出：“不能始终一致地做同一件事是人性的弱点，因而高度纪律性的方法学非常脆弱。”

要想让过程模型可用，要么必须提供实际可行的机制来维持必要的纪律，要么必须“宽容”地对待软件工程师。显而易见，“宽容”更易于被接受和保持，但是（正如 Cockburn 所认同）可能效率低下。正像人生中的大多数事情一样，必须权衡利弊。

关键概念

重构
Scrum
spike 解决方案
XP 故事

关键点

敏捷开发并不意味着不创建任何文件，但其仅在开发过程后期创建文件。

引述 敏捷：1，其他：0。

Tom DeMarco

5.1 什么是敏捷

在软件工作这个环境下，什么是敏捷？Ivar Jacobson[Jac02a] 给出一个非常有用的论述：

敏捷已经成为当今描述现代软件过程的时髦用词。每个人都是敏捷的。敏捷团队是能够适当响应变更的灵活团队。变更就是软件开发本身，软件构建有变更、团队成员在变更、使用新技术会带来变更，各种变更都会对开发的软件产品以及项目本身造成影响。我们必须接受“支持变更”的思想，它应当根植于软件开发中的每一件事中，因为它是软件的心脏与灵魂。敏捷团队意识到软件是团队中所有人共同开发完成的，这些人的个人技能和合作能力是项目成功的关键所在。

① 敏捷方法有时候也被称为轻量级方法或精简方法。

在 Jacobson 看来,普遍存在的变更是敏捷的基本动力,软件工程师必须加快步伐以适应 Jacobson 所描述的快速变更。

但是,敏捷不仅仅是有效地响应变更,它还包含着对本章开头的宣言中提及哲学观念的信奉。它鼓励能够使沟通(团队成员之间、技术和商务人员之间、软件工程师和经理之间)更便利的团队结构和协作态度。它强调可运行软件的快速交付而不那么看重中间产品(这并不总是好事情);它将客户作为开发团队的一部分开展工作,以消除持续、普遍存在于多数软件项目中的“区分我们和他们”的态度;它意识到在不确定的世界里计划是有局限性的,项目计划必须是可以灵活调整的。

敏捷可以应用于任何软件过程。但是,为了实现这一目标,非常重要的一点是:过程的设计应使项目团队适应于任务,并且使任务流水线化,在了解敏捷开发方法的流动性的前提下进行计划的制定,保留最重要的工作产品并使其保持简洁,强调这样一个增量交付策略——根据具体的产品类型和运行环境,尽可能快地将可工作的软件交付给客户。

建议 不要误解,以为敏捷会给你制定出解决方案的许可证。过程还是需要的,而规范是必不可少的。

5.2 敏捷及变更成本

软件开发的传统方法中(有几十年的开发经验作支持),变更成本随着计划的进展成非线性增长(图 5-1,黑色曲线)。这种方法在软件开发团队收集需求时(在项目的早期)相对容易适应变更。应用场景需要修改,功能表应该扩充,或者书面说明书需要编辑。这项工作的费用是最小的,所需的时间不会严重影响项目的结果。但是,如果我们在经过数月的开发时间之后将会怎么样?团队正在进行确认测试(也许是在项目后期的某个活动中),这时一个重要的利益相关者要求变更一个主要功能。这一变更需要对软件的体系结构设计进行修改,包括设计和构建三个新构件、修改另外五个构件、设计新的测试等。成本会迅速升级,为了保证变更不会引起非预期的副作用,所需的时间和费用都是非常可观的。

敏捷的拥护者(例如[Bec00]、[Amb04])认为,一个设计良好的敏捷过程“拉平”了变更曲线(图 5-1,灰色曲线),使软件开发团队在没有超常规的时间和费用影响的情况下,在软件项目后期能够适应各种变更。大家已经学习过,敏捷过程包括增量交付。当增量交付与其他敏捷实践结合时,例如连续单元测试及结对编程(在本章后面讨论),引起变更的费用会衰减。虽然关于拉平曲线程度的讨论仍然在进行,但是证据表明[Coc01a],变更成本显著降低。

引述 敏捷是动态的、针对特定内容的、主动应对变更及面向成长的。

Steven Goldman
et al.

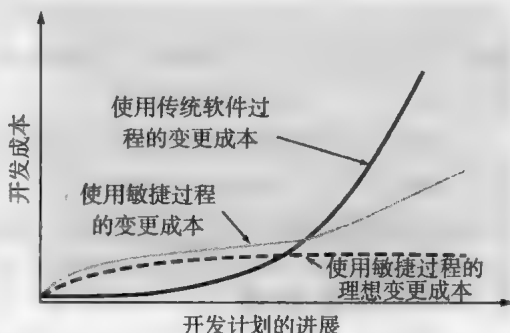


图 5-1 变更成本是开发时间的函数

关键点 敏捷过程能够降低变更的成本是因为软件产品以增量方式发布,而且在增量内部变更能得到较好的控制。

5.3 什么是敏捷过程

任何敏捷软件过程的特征都是以某种方式提出若干关键假设[Fow02],这些假设可适用于大多数软件项目。

1. 提前预测哪些需求是稳定的以及哪些需求会变更是非常困难的。同样, 预测项目进行客户优先级的变更也很困难。
2. 对很多软件来说, 设计和构建是交错进行的。也就是两种活动应当顺序开展以保证通过构建实施来验证设计模型, 而在通过构建验证之前很难估计应该设计到什么程度。
3. 分析、设计、构建和测试并不像我们所设想的那么容易预测(从制定计划的角度来看)。

网络资源 有关敏捷过程的综合文集可在以下网站找到: <http://www.agilemodeling.com>。

69

给出这三个假设, 同时也就提出一个重要的问题: 如何建立能解决不可预测性的过程? 正如前文所述, 答案就在于过程的可适应性(对于快速变更的项目和技术条件)。因此, 敏捷过程必须具有可适应性。

但是原地踏步式的连续适应性变更收效甚微, 因而, 敏捷软件过程必须增量地适应。为了达到这一目的, 敏捷团队需要客户的反馈(以做出正确的适应性改变), 可执行原型或部分实现的可运行系统是客户反馈的最有效媒介。因此, 应当使用增量式开发策略, 必须在很短的时间间隔内交付软件增量(可执行原型或部分实现的可运行系统)来适应(不可预测的)变更的步伐。这种迭代方法允许客户: 周期性地评价软件增量, 向软件项目组提出必要的反馈, 影响为适应反馈而对过程进行的适应性修改。

5.3.1 敏捷原则

敏捷联盟(参见[Agi03]、[Fow03])为希望达到敏捷的人们定义了12条原则:

1. 我们最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。
2. 即使在开发的后期, 也欢迎需求变更。敏捷过程利用变更为客户创造竞争优势。
3. 经常交付可运行软件, 交付的间隔可以从几个星期到几个月, 交付的时间间隔越短越好。
4. 在整个项目开发期间, 业务人员和开发人员必须天天都在一起工作。
5. 围绕有积极性的个人构建项目。给他们提供所需的环境和支持, 并且信任他们能够完成工作。
6. 在团队内部, 最富有效果和效率的信息传递方法是面对面交谈。
7. 可运行软件是进度的首要度量标准。
8. 敏捷过程提倡可持续的开发速度。责任人(sponsor)、开发者和用户应该能够长期保持稳定的开发速度。
9. 不断地关注优秀的技能和好的设计会增强敏捷能力。
10. 简单——使不必做的工作最大化的艺术——是必要的。
11. 最好的架构、需求和设计出自于自组织团队。
12. 每隔一定时间, 团队会反省如何才能更有效地工作, 并相应调整自己的行为。

关键点 尽管敏捷过程支持变更, 但检查变更的原因仍然是重要的。

建议 可运行的软件至关重要, 但不可忘记还必须使软件具有包括可靠性、可用性以及可维护性在内的各种质量属性。

70

并不是每一个敏捷过程模型都同等使用这12项原则, 一些模型可以选择忽略(或至少淡化)一项或多项原则的重要性。然而, 上述原则定义了一种敏捷精神, 这种精神贯穿于本章提出的每一个过程模型。

5.3.2 敏捷开发战略

与较为传统的软件工程过程相反,敏捷软件开发在优越性和适用性方面存在着许多(有时是激烈的)争议。在表达对敏捷拥护者阵营(“敏捷者”)的感想时,Jim Highsmith [Hig02a](开玩笑地)说明了他那颇为极端的观点:“传统方法学家陷入了误区,乐于生产完美的文档而不是满足业务需要的可运行系统。”而在表述对传统软件工程阵营的立场时,他则给出完全相反的观点(同样是玩笑性质的):“轻便级方法或者说敏捷方法学家是一群自以为了不起的黑客,他们妄图将其手中的软件玩具放大到企业级软件而制造出一系列轰动。”

建议 你不必在敏捷和软件工程之间做选择。只需定义敏捷的软件工程方法。

像所有的软件技术争论一样,这场方法学之争有滑向派别之战的危险。一旦争吵发生,理智的思考就消失了,信仰而不是事实主导着各种决定。

没有人反对敏捷。而真正的问题在于“什么是最佳实现途径”。同等重要的还有,如何构建满足用户当前需要的软件,同时展示具有能满足客户长期需求的扩展能力?

对这两个问题还没有绝对正确的答案。即便在敏捷学派内部,针对敏捷问题,也提出了很多有细微差异的过程模型(5.4节),每个模型内有一组“想法”(敏捷者们不愿称其为“工作任务”),显现出和传统软件工程的显著差异。同时,许多敏捷概念是从优秀的软件工程概念简单地修正而来的。归根结底:兼顾两派的优点则双方都能得到很多利益,而相互诽谤只会两败俱伤。

感兴趣的读者可以参看[Hig01]、[Hig02a]和[DeM02],这些文献针对很多重要技术和方针问题给出了饶有趣味的总结。

5.4 极限编程

为了更详尽地说明敏捷过程,在此提供一个称为极限编程(eXtreme Programming, XP)的论述,它是敏捷软件开发中使用最广泛的一种方法。虽然极限编程相关的思想和方法最早出现于20世纪80年代后期,但具有开创意义的著作由Kent Beck撰写[Bec04a]。近年来,XP的变种——工业XP(IXP)被提了出来,目标是在大型组织内部使用敏捷过程[Ker05]。

网络资源 屡获殊荣的包括XP处理模块的“过程模拟对策”可参见<http://www.ics.uci.edu/~emilyo/SimSE/download.html>。

5.4.1 极限编程过程

XP使用面向对象方法作为推荐的开发范型(附录2),它包含了策划、设计、编码和测试4个框架活动的规则和实践。图5-2描述了XP过程,并指出与各框架活动相关的关键概念和任务。以下段落将概括XP关键的活动。

提问 XP中的“故事”是什么?

策划。策划活动(也称为策划比赛)开始于倾听,这是一个需求收集活动,该活动要使XP团队技术成员理解软件的商业背景,充分感受要求的输出和主要特性及主要功能。倾听产生一系列“故事”(也称为用户故事),描述将要开发的软件所需要的输出、特性以及功能。每个故事(类似于第8章讲述的用例)由客户书写并置于一张索引卡上,客户根据对应特征或功能的综合业务价值标明故事的权值(即优先级)^①。XP团队成员评

网络资源 XP“策划比赛”可以参见<http://csis.pace.edu/~bergin/xxp/planninggame.html>。

① 一个故事的权值也可能取决于其他故事的存在。

估每一个故事，并给出以开发周数为度量单位的成本。如果某个故事的成本超过了3个开发周，则将请客户把该故事进一步细分，重新赋予权值并计算成本。重要的是应注意到新故事可以在任何时刻书写。

客户和XP团队共同决定如何将故事分组，并置于XP团队将要开发的下一个发行版本（下一个软件增量）中。一旦认可对下一个发布版本的基本承诺（就包括的故事、交付日期和其他项目事项），XP团队将以下述三种方式之一对有待开发的故事进行排序：（1）所有选定故事将在（几周之内）尽快实现；（2）具有最高价值的故事将移到进度表的前面并首先实现；（3）高风险故事将首先实现。

项目的第一个发行版本（也称为一个软件增量）交付之后，XP团队计算项目的速度。简而言之，项目速度是第一个发行版本中实现的客户故事个数。项目速度将用于：（1）帮助估计后续发行版本的发布日期和进度安排；（2）确定是否对整个开发项目中的所有故事有过分承诺。一旦发生过分承诺，则调整软件发行版本的内容或者改变最终交付日期。

在开发过程中，客户可以增加故事、改变故事的权值、分解或者去掉故事。接下来由XP团队重新考虑所有剩余的发行版本并相应修改计划。

设计。XP设计严格遵循KIS（Keep It Simple，保持简洁）原则，即使用简单的设计，而不是复杂的表述。另外，设计为故事提供恰好可实现的指导，而不鼓励额外功能性设计（因开发者假定以后会用到）^①。

XP鼓励使用CRC卡（第10章）作为在面向对象环境中考虑软件的有效机制。CRC（类-职责-协作者）卡确定和组织与当前软件增量相关的面向对象的类^②。XP团队使用类似于第10章描述的过程来管理设计工作。CRC卡也是作为XP过程一部分的唯一设计工作产品。

如果在某个故事设计中碰到困难，XP推荐立即建立这部分设计的可执行原型。实现并评估设计原型被称为spike解决方案。其目的是在真正实现开始时就降低风险，对可能存在设计问题的故事确认其最初的估计。

XP鼓励的重构既是构建技术又是设计技术，Fowler[Fow00]描述的重构如下：

重构是以不改变代码外部行为而改进其内部结构的方式来修改软件系统的过程。这是一种净化代码（并修改或简化内部设计）以尽可能减少引入错误的严格方法。实质上，重构就是在编码完成之后改进代码设计。

因为XP设计实际上不使用符号并且几乎不产生工作产品，如果有的话也只不过是生成除CRC卡和spike解决方案之外的工作产品，所以设计

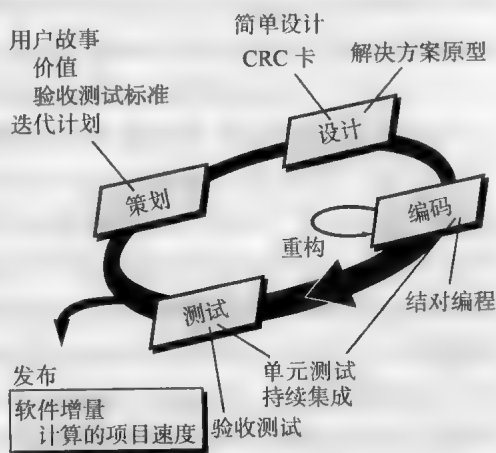


图 5-2 极限编程过程

关键点 项目速度是团队生产力的精妙度量。

建议 XP并不强调设计的重要性。这一点并不是所有人都认同。事实上，有些时候，应当强调设计。

网络资源 重构技术及其工具可在网站 www.refactoring.com 找到。

关键点 重构是以不改变其外部功能或行为而改进设计（或源代码）的内部结构。

① 虽然复杂的设计表示和术语可以加以简化，但每种软件工程方法都应遵循这些设计指导。

② 面向对象的类将在本书附录2、第10章中讨论，并贯穿本书的第二部分。

会被当作是可以并且应当在构建过程中连续修改的临时人工制品。重构的目的是控制那些“可以根本改进设计”的小的设计变更所要进行的修改 [Fow00]。然而应当注意的是，重构所需的工作量随着应用软件规模的增长而急剧增长。

XP 的中心观念是设计可以在编码开始前后同时进行，重构意味着设计随着系统的构建而连续进行。实际上，构建活动本身将给 XP 团队提供关于如何改进设计的指导。

编码。XP 推荐在故事开发和初步设计完成之后，团队不是直接开始编码，而是开发一系列用于检测本次（软件增量）^①发布的包括所有故事的单元测试，一旦建立起单元测试^②，开发者就更能集中精力于必须实现的内容以通过单元测试。不需要加任何额外的东西（KIS，保持简洁）。一旦编码完成，就可以立即完成单元测试，从而向开发者提供即时反馈。

结对编程。XP 建议两个人面对同一台计算机共同为一个故事开发代码。这一方案提供了实时解决问题（两个人总比一个人强）和实时质量保证的机制（在代码写出后及时得到复审），同时也使得开发者能集中精力于手头的问题。实施过程中，不同成员担任的角色略有不同，例如，一名成员考虑设计特定部分的编码细节，而另一名成员确保编码遵循特定的标准（XP 所要求的那些），或者确保故事相关的代码满足已开发的单元测试，并根据故事进行验证。^③

当结对的两人完成其工作后，他们所开发代码将与其他人的工作集成起来。有些情况下，这种集成作为集成团队的日常工作实施。还有一些情况下，结对者自己负责集成，这种“连续集成”策略有助于避免兼容性和接口问题，建立能及早发现错误的“冒烟测试”环境（第 22 章）。

测试。所建立的单元测试应当使用一个可以自动实施的框架（因此易于执行并可重复），这种方式支持每当代码修改（会经常发生，为 XP 提供重构理论）之后即时的回归测试策略（第 22 章）。

一旦将个人的单元测试组织到一个“通用测试集”[Wel99]，便每天都可以进行系统的集成和确认测试。这可以为 XP 团队提供连续的进展指示，也可在一旦发生问题的时候及早提出预警。Wells[Wel99]指出，“每几个小时修改一些小问题，要比仅仅在最后截止期之前修正大问题要节省时间。”

XP 验收测试也称为客户测试，由客户规定技术条件，并且着眼于客户可见的、可评审的系统级的特性和功能，验收测试根据本次软件发布中所实现的用户故事而确定。

网络资源 XP 的有用信息见于 www.xprogr amming.com。

提问 什么是结对编程？

建议 许多软件团队都是由个人构成的。必须努力改变这种文化，应该说结对编程很奏效。

提问 单元测试如何在 XP 中使用？

关键点 XP 验收测试是由用户故事驱动的。

5.4.2 工业极限编程

Joshua Kerievsky[Ker05] 这样描述工业极限编程（IXP）：“IXP 是 XP 的一种有机进化。它由 XP 的最低限要求、以客户为中心和测试驱动精神组成。IXP 与原来 XP 的主要差别在

① 这种方法类似于开始学习之前先了解考题。通过只集中注意力在将要回答的问题上，这使得研究要容易得多。

② 第 22 章详细地讨论了单元测试，集中于单个软件构件，检查构件接口、数据结构及功能，其目的在于发现构件内的错误。

③ 结对编程已在整个软件界普及，关于该主题，《华尔街日报》[Wal12] 用头版进行了报道。

于其管理具有更大的包容性，它扩大了用户角色，升级了技术实践。”IXP合并了六个新实践，这些新实践的设计是为了有助于确保在大型组织内的一些重要项目中，XP工作能够成功地实施。

准备评估。IXP团队确定该项目社区的所有成员（例如利益相关者、开发者、管理者）是否都准备就绪，是否建立了合适的环境，以及是否理解所涉及的技术水平。

项目社区。IXP团队确定人员及其所具有的技能是否合适，是否针对该项目已进行了阶段性培训。该“社区”包括技术专家和其他利益相关者。

项目特许。IXP团队通过对项目本身进行评估来确定对于项目的合适的商业调整是否存在，以及是否可以进一步深化组织机构的整体目标和目的。

测试驱动管理。IXP团队建立一系列可测量的“目标”[Ker05]，以评估迄今为止的进展情况，然后定义一些机制来确定是否已经实现了这些目标。

回顾。IXP团队在一个软件增量交付之后要实施特定的技术评审（第20章）。这种评审称为回顾，评审通过软件增量或者全部软件的发布过程复查“问题、事件以及经验教训”[Ker05]。

持续学习。鼓励（可以激励）XP团队的成员去学习新的方法和技术，从而获得高质量的软件产品。

除了以上讨论的六个新实践，IXP还修改了大量已有的XP实践，重新定义了特定的角色和职责，使他们更适合大型组织的重大项目。对于IXP的进一步讨论，请访问网站 <http://industrialxp.org>。

提问 为了生成IXP，在XP上附加了哪些新的实践？

引述 能力是你能够做什么，激励决定了你做什么，而态度决定了你做得怎样。

Lou Holtz

SafeHome 考虑敏捷软件开发

[场景] Doug Miller的办公室。

[人物] Doug Miller，软件经理；Jamie Lazor和Vinod Raman，软件团队成员。

[对话]

（敲门，Jamie和Vinod来到Doug的办公室。）

Jamie: Doug，有时间吗？

Doug: 当然，Jamie，什么事？

Jamie: 我们考虑过昨天讨论的过程了……就是我们打算为这个新的SafeHome项目选什么过程。

Doug: 哦？

Vinod: 我和在其他公司的一位朋友聊，他告诉了我极限编程。那是一种敏捷过程模型，听说过吗？

Doug: 听说过，有好也有坏。

Jamie: 对，看起来很适合我们。可以使软件开发更快，用结对编程来达到实时质量检查……我想这一定很酷。

Doug: 它确实有很多实实在在的好主意。比如，我喜欢其中的结对编程概念，还有利益相关者参加项目组的想法。

Jamie: 哦？你是说市场部将和项目组一起工作？

Doug (点头): 他们也是利益相关者，不是吗？

Jamie: 哇，他们会每隔5分钟就提出一些变更。

Vinod: 不要紧。我的朋友说XP项目有包容变更的方法。

Doug: 所以你俩认为我们应当使用XP？

Jamie：绝对值得考虑。

Doug：我同意。既然我们选择了增量模型方法，那就没有理由不利用 XP 带来的好处。

Vinod：Doug，刚才你说“有好处也有坏处”，坏处是什么？

Doug：我不喜欢 XP 不重视分析和设计……简而言之就是直接编码。（团队成员相视而笑。）

Doug：那你们同意用 XP 方法吗？

Jamie（代表二人说）：老板，我们干的就是编码！

Doug（大笑）：没错，但我希望看到你花少量时间编码和重新编码，而花多一点时间分析我们应当做什么并设计一个可用系统。

Vinod：或许我们可以二者兼用，带有一定纪律性的敏捷。

Doug：我想我们能行，Vinod，实际上我坚信这样的做法没问题。

5.5 其他敏捷过程模型

软件工程的历史是由散乱着的几十个废弃的过程描述和方法学、建模方法和表示法、工具以及技术所构成，每一个都是轰轰烈烈地冒出来，接着又被新的且（期望是）更好的所替代。随着敏捷过程模型的大范围推广（每一种模型都在争取得到软件开发界的认可），敏捷运动正在遵循着同样的历史步伐^①。

就像前一节提到的，在所有敏捷过程模型中使用最广泛的就是 XP。但是提出的许多其他敏捷过程模型也在整个行业中使用。在本节中，我们简要概述了四种常见的敏捷方法：Scrum、DSSD、敏捷建模（AM）以及敏捷统一过程（AUP）。

引述 我们这个专业实施方法论就如同 14 岁的少年穿衣服一样，还不成熟。

Stephen Hawrysh,
Jim Ruprecht

5.5.1 Scrum

Scrum（得名于橄榄球比赛^②）是 Jeff Sutherland 和他的团队在 20 世纪 90 年代早期发展的一种敏捷过程模型，近年来，Schwaber 和 Beedle 对其做了进一步的发展 [Sch01b]。

Scrum 原则与敏捷宣言是一致的，应用 Scrum 原则指导过程中的开发活动，过程由“需求、分析、设计、演化和交付”等框架性活动组成。每一个框架活动中，发生于一个过程模式（在以下段落中讨论）中的工作任务称为一个冲刺（sprint）。冲刺中进行的工作（每一个框架活动中冲刺的数目根据产品复杂度和规模大小而有所不同）适应于当前的问题，由 Scrum 团队规定并常常进行实时修改。Scrum 过程的全局流程如图 5-3 所示。

网络资源 关于 Scrum 的有用信息和资源见于 www.controlch.aos.com。

Scrum 强调使用一组“软件过程模式”[Noy02]，这些过程模式被证实在时间紧张的需求变更的和业务关键的项目中是有效的。每一个过程模式定义一系列开发活动。

① 这并不是坏事。在某个模型或方法被当作事实上的标准之前，它都在尽力争取软件工程师群体的人心。最终胜利者将发展成为最佳实践，而失败者将销声匿迹或是被融入取胜的模型。

② 一组球员围着球排列成圆圈，共同努力（有时具有暴力性）地想要带球扑地。

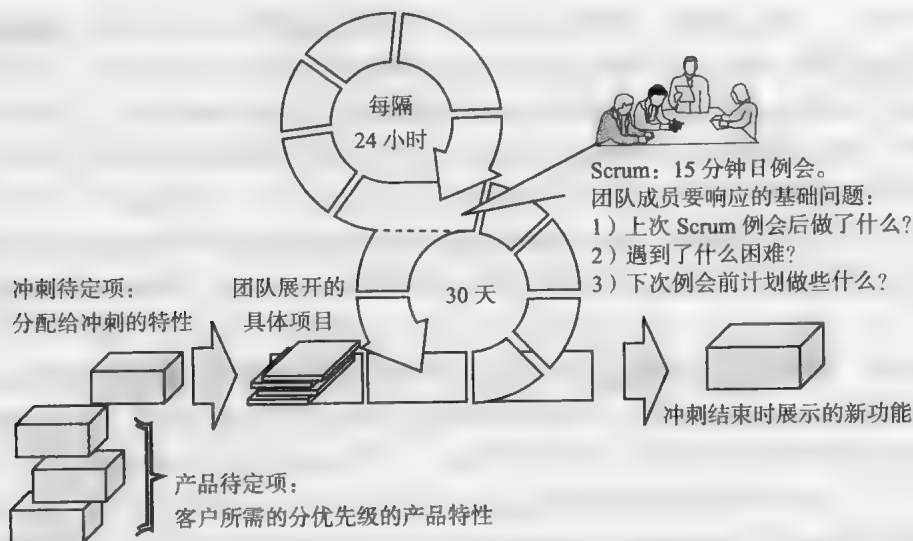


图 5-3 Scrum 过程流

待定项 (backlog)——一个能为用户提供商业价值的项目需求或特性的优先级列表。待定项中可以随时加入新项 (这就是变更的引入)。产品经理根据需要评估待定项并修改优先级。

冲刺 (sprint)——由一些工作单元组成, 这些工作单元是达到待定项中定义的需求所必需的, 并且必须能在预定的时间段 (time-box[⊖]) 内 (一般情况下为 30 天) 完成。冲刺过程中不允许有变更 (例如积压工作项)。因此, 冲刺给开发团队成员的工作提供了短期但稳定的环境。

Scrum 例会——Scrum 团队每天召开的短会 (一般情况为 15 分钟), 会上所有成员要回答三个问题 [Noy02]:

- 上次例会后做了什么?
- 遇到了什么困难?
- 下次例会前计划做些什么?

团队领导 (也称为 Scrum 主持人)主持会议并评价每个团队成员的表现。Scrum 会议帮助团队尽早发现潜在的问题。同时, 每日例会能够促进“知识社会化交流” [Bee99] 以及自我组织团队的建设。

演示——向客户交付软件增量, 为客户演示所实现的功能并由客户对其进行评价。需提醒的很重要一点是, 演示不需要包含计划内的所有功能, 但是规定该时间段内的可交付功能必须完成。

Beedle 和他的同事 [Bee99] 在他们所发表的关于这些模式的综合讨论中提到: “Scrum 预先假定混乱的存在……”。Scrum 过程模式保证软件开发团队在无法消除不确定的世界里能成功地工作。

5.5.2 动态系统开发方法

动态系统开发方法 (Dynamic System Development Method, DSDM) [Sta97] 是一种敏

⊖ Time-box 是一个项目管理术语 (见本书第四部分), 指的是分配给完成某一任务的时间段。

关键点 Scrum
由含有以下内容的一组过程模式构成: 强调项目优先次序、分离的工作单元、沟通以及频繁的客户反馈。

79

捷软件开发方法,该方法提供一种框架,使其“通过在可控项目环境中使用增量原型开发模式以完全满足对时间有约束的系统的构建和维护”[CCS02]。DSDM 建议借用修改版 Pareto (佩瑞多) 原则的哲学观念。这种情况下,如果交付整个应用系统需用 100% 时间,那么 80% 的应用系统可以用 20% 的时间交付。

DSDM 使用迭代软件过程,每一个迭代都遵循 80% 原则,即每个增量只完成能够保证顺利进入下一增量的工作,剩余的细节则可以在知道更多业务需求或者提出并同意变更之后完成。

DSDM 协会 (www.dsdm.org) 是一个由一些共同充当 DSDM 方法管理者的成员公司所组成的全球性组织。协会定义了一个称为 DSDM 生命周期的敏捷过程模型。该生命周期以建立基本的业务需求和约束的可行性研究开始,之后是识别功能和信息需求的业务研究。DSDM 定义了以下 3 个不同的迭代周期。

功能模型迭代——为客户开发一系列可证明其功能的增量原型(注意:所有 DSDM 原型都倾向于逐渐演化为可交付的应用系统)。这一迭代周期的意图是在用户使用原型系统时引导出反馈信息以获取补充的需求。

设计和构建迭代——在功能模型迭代中,重新构建原型以确保每一个原型都以工程化方式实现,并能为最终用户提供可操作的业务价值。有些情况下,功能模型迭代、设计和构建迭代可同步进行。

实现——将最终软件增量(一个可操作的原型)置于运行环境中。应当注意:(1) 增量不见得 100% 完成;(2) 增量置于运行环境以后可能需要变更。在这两种情况下,DSDM 开发转向功能模型迭代活动继续进行。

DSDM 和 XP (5.4 节) 可以结合使用,这种组合方法用具体实践 (XP) 定义固定的过程模型 (DSDM 生命周期),这些具体实践是构建软件增量所必需的。

5.5.3 敏捷建模

很多情况下,软件工程师必须构建大型的、业务关键型的系统,这种系统的范围和复杂性必须通过建模方式保证以下事项:(1) 所有参与者可以更好地理解要做什么;(2) 有效地将问题分解给要解决它的人;(3) 对正在设计和构建的系统质量进行评估。但在某些情况下,管理所需的符号量可能是艰巨的,艰巨性在于所建议采用模型形式化的程度、用于大型项目时模型的大小和变更发生时维护的难度。软件工程建模的敏捷方法能否为解决这些问题提供可选方案呢?

在敏捷建模官方网站上,Scott Ambler[Amb02a] 用以下方式描述敏捷建模 (Agile Modeling, AM):

AM 是一种基于实践的方法学,用于对基于软件的系统实施有效建模和文档编制。在软件开发项目中,AM 是可以有效并以轻量级方式用于软件建模的标准、原则和实践。由于敏捷模型只是大致完善,而不要求完美,因此敏捷模型比传统的模型更有效。

AM 采纳了与敏捷宣言一致的全部标准。敏捷建模的指导思想认为,敏捷团队必须有做出决定的勇气,哪怕这些决定可能否决当前的设计并导致重新构建。敏捷团队也必须保持谦逊作风,应当意识到技术不能解决所有问题,要虚心尊重并采纳业务专家和其他利益相关者

网络资源 有关 DSDM 的有用资源见 www.dsdm.org。

关键点 DSDM 是过程框架,可以采用另一种敏捷方法的策略,如 XP。

网络资源 有关敏捷建模更为全面的信息见于 www.agilemodeling.com。

80

的意见。

虽然 AM 提出一系列的“核心”和“补充”建模原则，但其中独具特色的是 [Amb02a] 以下原则。

有目的模型。在构建模型之前，使用 AM 的开发者心中应当有明确的目标（如与客户沟通信息，或有助于更好地理解软件的某些方面），一旦确定模型的目标，则该用哪种类型的表达方式以及所需要的具体细节程度都是显而易见的。

使用多个模型。描述软件可以使用多种不同的模型和表示法，大多数项目只用到其中很小的部分就够了。AM 建议从需要的角度看，每一种模型应当表达系统的不同侧面，并且应使用能够为预期读者提供价值的那些模型。

轻装上阵。随着软件工程工作的进展，只保留那些能提供长期有价值的模型，抛弃其余的模型。保留下来的每一个工作产品都必须随着变更而进行维护，这些描述工作将使整个团队进度变慢。Ambler[Amb02a] 提示说，每次决定保留一个模型，你都要在以抽象方式使用信息的便利性与敏捷性方面做权衡（即团队内部、团队与利益相关者增强沟通）。

内容重于表述形式。建模应当向预期的读者分享重要的信息。一个有用内容很少但语法完美的模型不如一个带有缺陷但能向读者提供有用内容的模型更有价值。

理解模型及工具。理解每一个模型及其构建工具的优缺点。

适应本地需要。建模方法应该适应敏捷团队的需要。

当前软件工程领域大都已采用了统一建模语言（UML）^①作为首选的方法来表示分析和设计模型。统一过程（第4章）已经为 UML 应用提供了一个框架。Scott Ambler[Amb06] 已经开发出集成了其敏捷建模原理的 UP 的简单版本。

5.5.4 敏捷统一过程

敏捷统一过程（Agile Unified Process, AUP）采用了一个“在大型上连续”以及“在小型上迭代”[Amb06] 的原理来构建基于计算机的系统。采用经典 UP 阶段性活动——起始、细化、构建和转换——AUP 提供了一系列活动（例如软件工程活动的一个线性序列），能够使团队为软件项目构想出一个全面的过程流。然而，在每一个活动里，一个团队迭代使用敏捷，并且将有意义的软件增量尽可能快地交付给最终用户。每个 AUP 迭代执行以下活动 [Amb06]。

- 建模。建立对商业和问题域的 UML 表述。然而，为了保持敏捷，这些模型应当“足够好”[Amb06]，以使团队继续前进。
- 实现。将模型翻译成源代码。
- 测试。像 XP 一样，团队设计和执行一系列的测试来发现错误以保证源代码满足需求。
- 部署。就像第3章中讨论过的一般过程活动，这部分的部署重点仍然是对软件增量的交付以及获取最终用户的反馈信息。

引述 几天前我去药店打算买些感冒药，可是不容易啊。看到有太多的药可选，往前走走，是一种速效的，还有一种是长效的。究竟选哪个好呢？是图眼前还是图长远呢？

Jerry Seinfeld

建议 对所有软件工作而言，“轻装”都是适合的指导思想。我们只需要那些有价值的模型，不要多，也不要少。

① 在本书附录1中提供了UML的引论。

- 配置及项目管理。在 AUP 中，配置管理（第 29 章）着眼于变更管理、风险管理以及对开发团队的任一常效产品^①的控制。项目管理追踪和控制开发团队的工作进展并协调团队活动。

- 环境管理。环境管理协调过程基础设施，包括标准、工具以及适用于开发团队的支持技术。

虽然 AUP 与统一建模语言有历史上和技术上的关联，但是很重要的一点必须注意，UML 模型可以与本章所讲的任一敏捷过程模型相结合。

软件工具 敏捷开发

[目标] 敏捷开发工具的目标是辅助敏捷开发的一个或多个方面，强调便利地快速构建可执行软件。这些工具也可以用于惯例（传统）过程模型（第 4 章）的开发。

[机制] 工具的机制各不相同。通常，敏捷工具集包括项目计划、用例开发和需求收集、快速设计、代码生成和测试的自动支持。

[代表性工具]^②

注意：由于敏捷开发是一个热门话题，因此大多数软件工具供应商都声称出售支持敏捷方法的工具。下面的工具具有的特

性对敏捷项目非常有用。

- OnTime。由 Axosoft 开发（www.axosoft.com），提供对各种技术活动的敏捷过程管理支持。
- Ideogramic UML。由 Ideogramic 开发（<http://ideogramic-uml.software.informer.com>），是特别为敏捷过程开发的 UML 工具集。
- Together Tool Set。由 Borland 销售（www.borland.com），提供支持 XP 和其他敏捷过程中许多技术活动的工具包。

5.6 敏捷过程工具集

敏捷哲学的拥护者指出，自动软件工具（例如设计工具）应当被看作是对开发团队活动小小的补充，而不是团队成功的关键。然而，Alistair Cockburn[Coc04] 建议，工具是有益处的，“敏捷团队强调使用工具可以达到快速理解的目的。有些工具是社会性的，甚至开始于租赁阶段。有些工具是技术性的，可以帮助使用者团队模拟物理现状。很多工具是物理性的，允许人们在工作场所操作这些工具。”

协作和沟通“工具”通常技术含量较低，并且与可以提供信息以及协调敏捷开发人员的任何机制相结合（“这些机制可以是物理上的近距离性、白板、海报、索引卡以及可粘贴的留言条”[Coc04] 或现代社会网络技术）。积极的沟通是通过团队能动性获得的（例如结对编程），而被动的沟通是通过“信息辐射体”实现的（例如，一台平面显示器可以显示一个增

关键点 这里所提到的敏捷过程“工具集”更多地是从人员方面而不是从技术方面支持敏捷过程。

① 常效工作产品指的是被管理的团队在不同阶段开发的模型、文档或测试用例，这些产品在软件增量交付后并不废弃。

② 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

量不同组件的全部状态)。项目管理工具降低了甘特 (Gantt) 图的重要性, 人们使用挣值图 (earned value chart) 来取代甘特图或“所创建的测试与所通过的测试对比图……还有其他工具可用来优化敏捷团队工作的环境 (例如更有效的会议区), 通过培养社交互动 (例如配置团队)、物理设备 (例如电子白色书写板)、过程增强 (例如结对编程或时间段)” [Coc04] 等提高团队文化。

所有这些都是工具吗? 如果它们能够促进敏捷团队成员的工作并提高最终产品的质量, 那么它们就是工具。

5.7 小结

在现代经济中, 市场条件变化十分迅速, 客户和最终用户的要求在更新, 新一轮竞争威胁会没有任何征兆地出现。从业者必须使软件工作保持敏捷——要定义灵活机动、有适应能力和精益的过程以适应现代商务的需求。

软件工程的敏捷理念强调四个关键问题: 自我组织团队对所开展工作具有控制力的重要性; 团队成员之间以及开发参与者与客户之间的交流与合作; 对“变更代表机遇”的认识; 强调快速软件交付以让客户满意。敏捷过程模型能解决上述这些问题。

极限编程 (XP) 是应用最广泛的敏捷过程。按照策划、设计、编码和测试四个框架活动组织, XP 提出一系列新颖和有力的技术, 以保证敏捷团队创建的能体现利益相关者指定优先级特性和功能的软件能频繁发布。

其他敏捷过程模型也强调人员合作和团队自组织, 只是定义了自己的框架活动, 选择不同的侧重点。例如, Scrum 强调一系列软件过程模式的使用, 这些模式已被证实对时间紧迫、需求变更和业务关键的项目非常有效。每一个过程模式定义一系列开发任务并允许 Scrum 团队以适应其项目要求的方式构建过程。动态系统开发方法 (DSDM) 倡导时间调度的使用, 认为进入下一次增量开发前只需对每一个软件增量做足够的工作。敏捷建模 (AM) 表明建模对于所有的系统都是必要的, 但是模型的复杂度、类型和规模必须根据所构建的软件来调节。敏捷统一过程 (AUP) 采用“全局串行”以及“局部迭代”的原则来构建软件。

84

习题与思考题

- 5.1 重新阅读本章开头的“敏捷软件开发宣言”, 你能否想出一种情况, 此时四个权值中的一个或多个将使软件开发团队陷入麻烦?
- 5.2 用自己的话描述 (用于软件项目的) 敏捷性。
- 5.3 为什么迭代过程更容易管理变更? 是不是本章所讨论的每一个敏捷过程都是迭代的? 只用一次迭代就能完成项目的敏捷过程是否存在? 解释你的答案。
- 5.4 是否每一个敏捷过程都可以用第3章所提及的通用框架活动来描述? 建一张表, 将通用活动和每个敏捷过程所定义的活动对应起来。
- 5.5 试着再加上一条“敏捷性原则”, 以便帮助软件工程团队更具有机动性。
- 5.6 选择5.3.1节提到的一条敏捷性原则, 讨论本章所描述的各过程模型是否符合该原则。(注意: 我们只是给出了这些过程模型的概述, 因此, 确定一个或多个模型是否符合某个原则或许不可能, 需要做更多的研究 (对这个问题并不需要)。)
- 5.7 为什么需求变更这么大? 人们终究无法确定他们想要什么吗?
- 5.8 大多数敏捷过程模型推荐面对面交流。然而, 现在软件开发团队成员及其客户在地理上是相互分散的。你是否认为应该避免这种地理上的分散? 能否想出一个办法克服这个问题?

- 5.9 写出一个 XP 用户故事，描述适用于大部分网页浏览器的“最喜欢的地方”或“最喜欢的事物”特性。
- 5.10 什么是 XP 的 spike 解决方案？
- 5.11 用自己的话描述 XP 的重构和结对编程的概念。
- 5.12 利用第 3 章介绍的过程模式模板，为 5.5.1 节提出的任意一个 Scrum 模式开发出过程模式。
- 5.13 访问敏捷建模官方网站，列出所有核心的和补充性的 AM 原则。
- 5.14 5.6 节中给出的工具集为敏捷方法的“软”方面提供了很多支持。由于交流非常重要，因此请推荐一种实际工具集，用来增强敏捷团队中利益相关者之间的交流。

扩展阅读与信息资源

敏捷软件开发的全部理论和基本原则在本章提供的很多参考书中都有深入的论述。另外，在 Pichler (《Agile Project Management with Scrum: Creating Products that Customers Love》，Addison-Wesley, 2010)、Highsmith (《Agile Project Management: Creating Innovative Products》，2nd ed. Addison-Wesley, 2009)、Shore 和 Chromatic (《The Art of Agile Development》，O'Reilly Media, 2008)、Hunt (《Agile Software Construction》，Springer, 2005) 以及 Carmichael 和 Haywood (《Better Software Faster》，Prentice Hall, 2002) 的书中给出了关于该主题的有力讨论。Aguanno (《Managing Agile Projects》，Multi-media Publications, 2005) 和 Larman (《Agile and Iterative Development: A Manager's Guide》，Addison-Wesley, 2003) 介绍了关于管理的概述及项目管理问题的思考。Highsmith (《Agile Software Development Ecosystems》，Addison-Wesley, 2002) 介绍了关于敏捷原则、过程和实践的调查。Booch 及其同事 (《Balancing Agility and Discipline》，Addison-Wesley, 2004) 给出了关于平衡敏捷性和规范性的颇有价值的讨论。

Martin (《Clean Code : A Handbook of Agile Software Craftsmanship》，Prentice-Hall, 2009) 指出了在敏捷软件工程环境下开发“干净代码”所需的敏捷原则、模式和实践。Leffingwell (《Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise》，Addison-Wesley, 2011 和《Scaling Software Agility : Best Practices for Large Enterprises》，Addison-Wesley, 2007) 讨论了在大型项目中扩大敏捷实践的策略。Lippert 和 Rook (《Refactoring in Large Software Projects : Performing Complex Restructurings Successfully》，Wiley, 2006) 讨论了在大型、复杂系统中重构的使用。Stamelos 和 Sfetsos (《Agile Software Development Quality Assurance》，IGI Global, 2007) 讨论了符合敏捷理论的 SQA 技术。

在过去的 10 年间，已经有很多书描述了极限编程。Beck (《Extreme Programming Explained : Embrace Change》，2nd ed., Addison-Wesley, 2004) 的书依然是本主题的权威著作。另外，Jeffries 及其同事 (《Extreme Programming Installed》，Addison-Wesley, 2000)、Succi 和 Marchesi (《Extreme Programming Examined》，Addison-Wesley, 2001)、Newkirk 和 Martin (《Extreme Programming in Practice》，Addison-Wesley, 2001) 以及 Auer 及其同事 (《Extreme Programming Applied: Play to Win》，Addison-Wesley, 2001) 的著作中，就如何更好地应用 XP 给出了关键的有指导意义的讨论。McBreen (《Questioning Extreme Programming》，Addison-Wesley, 2003) 则以挑剔的眼光审视 XP，定义了其何时、何地更为适用。对结对编程的深入考虑可阅读 McBreen (《Pair Programming Illuminated》，Addison-Wesley, 2003)。

Kohut (《Professional Agile Development Process: Real World Development Using SCRUM》，Wrox, 2013)、Rubin (《Essential Scrum: A Practical Guide to the Most Popular Agile Process》，

Addison-Wesley, 2012)、Larman 和 Vodde (《Scaling Lean and Agile Development: Thinking and Organizational Tools for Large Scale Scrum》, Addison-Wesley, 2008) 以及 Schwaber (《The Enterprise and Scrum》, Microsoft Press, 2007) 讨论了对主营业务产生影响的项目中使用 Scrum。Cohn (《Succeeding with Agile》, Addison-Wesley, 2009) 以及 Schwaber 和 Beedle (《Agile Software Development with SCRUM》, Prentice-Hall, 2001) 发表了对 Scrum 方法的深入探讨。DSDM Consortium (《DSDM: Business Focused Development》, 2nd ed., Pearson Education, 2003) 和 Stapleton (《DSDM: The Method in Practice》, Addison-Wesley, 1997) 的书提供了关于 DSDM 方法的有价值的论述。

Ambler 和 Lines (《Disciplined Agile Delivery: A Practitioner's Guide to Agile Delivery in the Enterprise》, IBM Press, 2012) 以及 Poppendieck (《Lean Development: An Agile Toolkit for Software Development Managers》, Addison-Wesley, 2003) 为管理和控制敏捷工程提供了指导。Ambler 和 Jeffries (《Agile Modeling》, Wiley, 2002) 深入探讨了敏捷建模。

可以从网上获得关于敏捷软件开发的更多信息资源, 有关敏捷过程的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

软件工程的人员方面

要点浏览

概念：我们都在努力学习最新的编程语言、最好的新型设计方法、最流行的敏捷过程或最新发布的热门软件工具。但说到底，是人在开发计算机软件。因此，在软件工程中，人对一个项目的成功所起的作用和那些最新最好的技术是一样的。

人员：软件工程师工作是由个人和团队完成的。在某些情况下，个人要承担大部分的责任，但大多数行业级软件要由团队完成。

重要性：只有团队的活力恰到好处，软件团队才能成功。软件工程师有时会给人不好相处的印象。但实际上，在要构建的产品中，团队里的工程师与同事及其他利益相关者进行良好合作是非常必要的。

步骤：首先，你要了解并不断积累一个成功的软件工程师应具备的个人特质。然后，你需要提升软件工程师工作中应具备的综合心理素质，这样才能在进行项目时少走弯路，避免失误。而且，你要理解软件团队的结构和动态，因为基于团队的软件工程在行业中很常见。最后，你需要重视社交媒体、云端和其他协作工具的影响力。

工作产品：对人员、过程和最终产品有更好的洞察力。

质量保证措施：花时间去观察成功的软件工程师是如何工作的，并调整自己的方法来利用他们所展现的优点。

在《IEEE 软件》的一期特刊中，客邀编辑们 [deS09] 做出如下评论：

软件工程包括大量可以改善软件开发过程和最终产品的技术、工具和方法。技术不断进步并产生了很多鼓舞人心的成果。然而，软件不单是用适合的技术方案解决不适合的技术手段而创造出的一种产品。软件由人开发、被人使用并支持人与人之间的互动。因此，人的特质、行为和合作是实际的软件开发的重心。

在本章之后的章节中，我们将讨论构建成功的软件产品所需的“技术、工具和方法”。但在此之前，我们有必要知道，如果没有技术娴熟并积极参与的人员，那么软件项目是不太可能成功的。

关键概念

敏捷团队
云计算
合作开发环境 (CDE)
全球化团队
有凝聚力的团队
心理学
角色
社交媒体
团队属性
团队结构
团队毒性
特性
XP 团队

6.1 软件工程师的特质

你想成为软件工程师吗？很显然，你需要掌握技术，学习并运用那些理解问题所需的技能，设计有效的解决方案，构造软件并努力测试以达到质量最优。你需要管理变更，与利益相关者沟通，并在合适的情况下使用合适的工具。这些我们都会在本书后面章节中用大量篇幅进行讨论。

但有些事和上述的这些同样重要——就是人的方面，掌握这些方面会让你成为卓有成效的软件工程师。Erdogmus[Erd09]指出软件工程师个人要想展现“非常专业的”行为，应具备七种特质。

一个卓有成效的软件工程师有个人责任感。这会让软件工程师去努力实现对同伴、其他利益相关者和管理者的承诺。为获得成功的结果，他会在需要的时候不遗余力地做他需要做的事情。

一个卓有成效的软件工程师对一些人的需求有敏锐的意识，这些人包括同团队的成员、对现存问题的软件解决方法有需求的利益相关者、掌控整个项目并能找到解决方法的管理者。他会观察人们工作的环境，并根据环境和人本身来调整自己的行为。

一个卓有成效的软件工程师是坦诚的。如果发现了有缺陷的设计，他会用诚实且有建设性的方式指出错误。即使被要求歪曲与进度、特点、性能、其他产品或项目特性有关的事实，他也会选择实事求是。

一个卓有成效的软件工程师会展现抗压能力。前面我们提到，软件工程师经常处在混乱的边缘。压力（以及由此产生的混乱）来自很多方面——需求和优先级的变更、要求苛刻的利益相关者或同伴、不切实际或者令人难以忍受的管理者。但一个卓有成效的软件工程师可以在不影响业绩的情况下很好地处理这些压力。

一个卓有成效的软件工程师有高度的公平感。他会乐于与同事分享荣誉，努力避免利益冲突并且绝不破坏他人劳动成果。

一个卓有成效的软件工程师注重细节。这并不意味着追求完美，而是说他会利用产品和项目已有的概括性标准（如性能、成本、质量），在日常工作基础上仔细思考，进而做出技术性决策。

最后，一个卓有成效的软件工程师是务实的。他知道软件工程不是要恪守教条的宗教信仰，而是可根据当下情景需要进行调整的科学规则。

引述 大多数优秀的程序员不是因为报酬或公众关注而做编程，而是因为兴趣。
Linus Torvalds

提问 一个高效率的软件工程师需要具备哪些个人特质？

6.2 软件工程心理学

在一篇有关软件工程心理学的学术论文中，Bill Curtis 和 Diane Walz[Cur90] 针对软件开发提出了一种分层的行为模式（图 6-1）。在个人层面，软件工程心理学注重待解决的问题、解决问题所需的技能以及在模型外层建立的限制内解决问题的动机。在团队和项目层面，团队能动性成为主要因素。在这一层面，成功是由团队结构和社会因素决定的。团队交流、合作和协调同单个团队成员的技能同等重要。在外部层面，有组织的行为控制着公司的行为及其对商业环境的应对方式。

在团队层面，Sawyer 和他的同事 [Saw08] 认为团队经常会制造虚拟的界线，这种界线会限制交流，因而会降低团队效率。他们提出了一组“跨界角色”，这能让软件团队成员在不同团队之间有效推进工作。下面这些角色要么是被特定指派的，要么是在工作中自然而然衍生出来的。

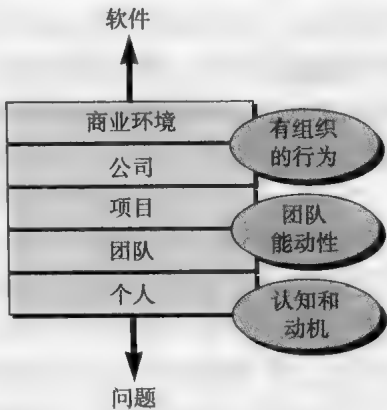


图 6-1 软件工程中的行为模式层（改自 [Cur90]）

- 外联络员——代表团队就时间和资源问题与外部的顾客谈判，并取得利益相关者的反馈。
- 侦察员——突破团队界线收集组织信息。“侦察活动包括：审视外部市场，寻求新技术，确定团队外界的相关活动，弄清潜在对手的活动。”[Saw08]
- 守护员——保护团队工作产品和其他信息产品。
- 安检员——把控利益相关者和他人向团队传送的信息。
- 协调员——注重横跨团队及组织内部的交流（如与组织内部的专家团队讨论特定的设计问题）。

提问 软件团队中的成员都会担任哪些角色？

6.3 软件团队

Tom DeMarco 和 Tim Lister[DeM98] 在他们的经典著作《Peopleware》中讨论了软件团队的凝聚力：

在商业领域内，我们经常使用团队这个词，把任何一组被派到一起工作的人称为一个“团队”。但是很多这样的组合并不像是团队。这些组合中的人对成功没有共同的认识，或者没有明确的团队精神。他们缺少的就是凝聚力。

提问 什么是“有凝聚力的”团队？

一个有凝聚力的团队中的人应该能强烈认识到整体比个体的简单相加更强大。

一旦团队凝聚起来，成功的可能性就会变大。整个团队会变得势不可当、坚不可摧……他们不需要传统的管理方式，当然也不需要外界的推动。他们本来就有动力。

DeMarco 和 Lister 认为有凝聚力的团队中的成员比其他人员生产力更高也更有动力。他们有共同的目标、共同的文化和，而且在大多数情况下，一种“精英意识”会让他们变得独特。

现在还没有能打造具有凝聚力团队的万无一失的方法。但高效的软件团队总是会显现一些特征^①。Miguel Carrasco[Car08] 认为高效的团队必须建立目标意识。比如，如果说团队成员认同团队的目标是开发可以转化产品类别的软件，并为此让公司一跃成为行业的领跑者，就可以说他们有很强的目标意识。高效的团队还必须要有参与意识，让每个成员都能感受到自己的技能得到了发挥，所做出的贡献是有价值的。

关键点 高效的软件团队是多样化的，是由具备目标意识、参与意识、信任意识和进步意识的人组成的。

高效的团队应该能培养信任意识。团队中的软件工程师应该相信同伴和其管理者的技术与能力。团队应该鼓励进步意识，定期审视软件工程方法并寻求改善途径。

最高效的团队是多样化的，由具备不同技能的人员组成。技术高超的工程师会与技术背景较弱但对利益相关者的需求更敏感的队员搭档。

但不是所有的团队都高效，也不是所有的团队都具有凝聚力。事实上，很多团队都在遭受着 Jackman[Jac98] 所说的“团队毒性”。她定义了五个“可能形成有害团队环境”的因素：（1）混乱的工作氛围；（2）会造成团队成员分裂的挫败；（3）“支离破碎或协调不当”的软件过程；（4）对软件团队中角色

提问 为什么团队会没有凝聚力？

① Bruce Tuckman 发现成功的团队在取得成果的工程中会经历四个阶段（形成，争执，规范，执行）(<http://www.realsoftwaredevelopment.com/7-key-attributes-of-high-performance-software-development-teams/>)。

的模糊定义；(5)“持续且重复性的失败”。

为避免混乱的工作环境，团队应获得工作所需的所有信息。一旦确定了主要目标，不到必要时刻就不轻易改变。为避免挫败，软件团队应该尽可能地对所做的决定负责。通过了解要完成的产品、完成工作的人员以及允许团队来选择过程模型，可以避免不当的软件过程（如不必要或过于繁重的任务，或者选择错误的工作产品）。团队自身要建立责任机制（技术评审^①是完成这一事项的好方法），在团队成员出现失误时找出正确的方法。最后，避免陷入失败氛围的关键是建立以团队为基础的反馈和问题解决技巧。

除了 Jackman 提到的五个毒性，软件团队还经常遇到团队成员特质不同的问题。有些团队成员性格外向，有些性格内向。有的成员会直观地收集信息，从各种事实中提取概括性观点。而有的成员会有序地处理信息，从已知数据中搜集和整理详尽的细节。有的成员在进行逻辑性、有序的讨论之后做决定。而有的凭直觉，更愿意凭“感觉”做决定。有的成员需要组织任务的详细进度安排，以使它们能够完成项目的某些部分。而有的团队希望有更加自发性的环境，也可以存在未定论的问题。有的成员会在里程碑日期之前很久就把工作完成，以免到时候有压力，而有的人会从最后时刻的紧迫感中受到激励。本节总结了对人的差异的认识以及其他指导规则，这些有助于更好地构建具有凝聚力的团队。

引述 不是每个组合都是一个团队，不是每个团队都是有效率的。
Glenn Parker

91

6.4 团队结构

“最佳”团队结构取决于组织的管理风格、团队组成人员的数量以及他们的技术水平，还有整体的问题难度。Mantei[Man81] 提出了一些在策划软件工程团队结构时应考虑的项目因素：(1) 需解决问题的难度；(2) 基于代码行或功能点^②的结果程序的“规模”；(3) 团队成员合作的时间（团队寿命）；(4) 问题可模块化的程度；(5) 所建系统的质量和可靠性；(6) 交付日期要求的严格程度；(7) 项目所需的社会化（交流）程度。

Constantine[Con93] 针对软件工程团队提出了四种“组织模式”：

1. 封闭模式组成的团队遵循传统的权力层级模式。这样的团队在建立与之前的成果十分相似的软件时能做得很好，但以封闭模式工作时创新性上相对较弱。
2. 随机模式组成的团队是松散的，并依靠团队成员的个人自发性。在需要创新和技术性突破时，这类团队可以做得很优秀。但是很难完成“有秩序的操作”。
3. 开放模式尝试组成一种团队，既具有封闭模式团队的可控性，还具有随机模式团队的创新性。成员们合作完成工作，并有丰富的交流和达成共识的决定，这些都是开放模式团队的特点。开放模式团队适合解决复杂的问题，但没有其他团队的效率高。
4. 同步模式组成的团队有赖于问题的自然区分，不需要很多的交流就可以将成员组织起来共同解决问题。

作为历史前鉴，最早的软件团队之一是被称为主程序员团队的封闭模

提问 选择软件团队的结构时应考虑哪些因素？

提问 定义软件团队的结构时有哪些选择？

引述 如果想逐渐变好，就要具有竞争力；如果想指数级变好，就要合作。
作者不详

① 有关技术评审的详细讨论见第20章。

② 代码行（Lines of Code, LOC）和功能点可测量电脑程序规模，见第33章。

92

式团队。这种结构最早由 Harlan Mills 提出, 由 Baker[Bak72] 建立。这类团队的核心包括: 一个高级工程师 (主程序员), 负责计划、协调并审查团队的所有技术活动; 技术人员 (通常 2~5 人), 进行分析和开发活动; 一名支持工程师, 协助高级工程师的工作, 为最小化项目持续性的损失, 有时可代替高级工程师。主程序员可以得到以下人员的服务支持: 一位或者多位专家 (如通信专家、数据库设计者)、支持人员 (如技术写作人员、文书人员) 和一个软件管理员。

对于主程序员团队结构, Constantine 的随机模式 [Con93] 观点认为, 对于具有创造独立性的团队, 其工作方法最好是创新的无序。尽管完成软件工作需要自由意识, 但是软件工程组织的核心目标必须是将创新活力运用于创建高效能团队。

SafeHome 团队结构

[场景] Doug Miller 的办公室, 优先致力于 SafeHome 软件项目。

[人物] Doug Miller (SafeHome 软件团队的管理者), Vinod Raman、Jamie Lazar 以及其他产品软件工程团队的成员。

[对话]

Doug: 你们有机会看一下市场部准备的有关 SafeHome 的基础信息。

Vinod (点了点头, 看着他的队友们): 是的, 但是我们遇到了很多问题。

Doug: 我们先不谈问题, 我想探讨一下怎样构建团队, 谁对什么问题负责……

Jamie: Doug, 我对敏捷理念很感兴趣。我

觉得我们可以成为一个自发组织的团队。

Vinod: 同意。鉴于时间有限, 不确定因素很多, 而且我们都很有能力 (笑), 这似乎是很好的方法。

Doug: 我没意见, 而且你们也知道怎么做。

Jamie (微笑并像在背诵什么一样说): 我们只做策略上的决定, 谁在什么时候做什么, 但是我们的任务是按时推出产品。

Vinod: 还要保证质量。

Doug: 很对。但是记住有一些限制。市场限制了需要制造出来的软件增量——当然是在与我们商讨的基础上。

Jamie: 然后呢?

6.5 敏捷团队

在过去的 10 年里, 敏捷软件开发 (第 5 章) 被认为放大了问题, 扰乱了软件项目工作。回顾一下, 敏捷理念支持: 客户满意且尽早的软件增量发布, 小型的充满动力的项目团队, 非正式方法, 最少的软件工程工作产品以及整体开发的简化。

6.5.1 通用敏捷团队

小型的并充满动力的项目团队也可称为敏捷团队, 他们具备前面章节中所讨论的成功软件项目团队的很多特征 (我们会在之后的章节谈到这些特征), 并且能够避免产生问题的很多毒素。但是, 敏捷理念强调个人 (团队成员) 通过团队合作可以加倍的能力, 这是团队成功的关键因素。Cockburn 和 Highsmith[Coc01a] 在他们的文章中谈道:

如果一个项目中的人员都足够优秀, 那么他们可以借助任意过程来完成任务。如果他们不够优秀, 那么也就没有什么过程能弥补他们的不足——即“人员胜过过程”。然而, 缺乏

关键点 敏捷团队是一个能自主计划和做出技术性决定的自发组织团队。

93

用户和决策支持也会扼杀一个项目——即“政策胜过人员”。如果不能得到足够的支持，优秀的人员也无法完成工作。

为了有效利用每个团队成员的能力，并完成项目工程过程中的高效合作，敏捷团队都是自组织的。一个自组织的团队不必保持单一的团队结构，而是综合运用6.2节中讨论的Constantine提出的随机、开放和同步模式。

很多敏捷过程模型（如Scrum）给予敏捷团队重要的自主性，允许团队自主做出完成工作必需的项目管理和技术决定。计划被保持在最低程度，团队可以选择自己的方式（如过程、方法、工具），仅受商业要求和组织标准的限制。在项目进行过程中，团队自发地及时将重点放在有益于项目特定点的个人能力上。为完成这项工作，敏捷团队可能每天都开例会，协商和同步当天必须要完成的事情。

基于例会期间获得的信息，团队将调整其方法以完成工作增量。随着时间的积累，持续的自发组织和合作可以促使团队完成软件增量工作。

6.5.2 XP 团队

作为极限编程（eXtreme Programming, XP）的一部分，Beck[Bec04a]定义了五项价值作为实施所有工作的基础——沟通、简单、反馈、勇气、尊重。任何一项价值都可以促成特定的XP活动、动作和任务。

为使敏捷团队和其他利益相关者达到有效的沟通（如建立所需的软件特性和功能），XP强调客户和开发者之间密切而非正式（口头）的合作，构建用作交流媒介的有效的隐喻^①，以便交流重要概念、获得持续反馈并避免冗长的文档。

为了达到简单，敏捷团队在设计时只是考虑当下需要而非长远需求。其目的是创建简单的设计，可以容易地用代码实现。如果必须要改进设计，以后可以对代码进行重构^②。

反馈来源于三种途径：所实现的软件本身、客户以及其他软件团队成员。通过设计和实行有效的测试策略（第22～26章），软件（通过测试结果）可以为敏捷团队提供反馈。团队可以利用单元测试作为其初始测试手段。每个类开发完成后，团队会根据其特定的功能开发单元测试来不断完善每个操作。向客户交付增量时，要用增量所实现的用户故事或者用例（第9章）进行验收测试。软件实现用例的输出、功能和行为的程度是一种形式的反馈。最终，在迭代计划中，新的需求会不断出现，团队可以就成本和进度的影响给客户id提供快速的反馈。

Beck[Bec04a]认为严格地坚持特定XP实践是需要勇气的。换个更好的说法是要有原则。比如说，设计经常会面临要考虑长远需求的巨大压力。大多数软件团队都妥协了，并辩称“为明天做设计”可以在长期发展中节省时间和精力。XP团队必须有原则（勇气），要为今天做设计，要认识到长远需求可能会发生意想不到的变更，因此会带来设计和实现代码的大量返工。

引述 集体所有体现了一种观点：工作成果是属于整个（敏捷）团队而非组成团队的个人的。

Jim Highsmith

建议 尽量保持简单，但需认识到持续的“重构”能获得可观的时间和资源。

94

引述 怎样在不费力的情况下构建优秀的软件？XP就是答案。

佚名

① 在XP中，隐喻是“任何人——客户、程序员和管理者——都能讲述系统如何工作的一个故事。”[Bec04a]。

② 重构是软件工程师在不改变设计（或源代码）的外部功能或行为的情况下改进其内部结构。实质上，重构可以用来改善设计或实现代码的效率、可读性或性能。

间以及间接地对软件本身的尊重。在成功发布软件增量时，团队对 XP 过程的重视就会油然而生。

6.6 社交媒体的影响

邮件、短信或者视频会议在软件工程工作中无处不在。但是这些交流机制不过是现代化替代品或面对面沟通机制的补充。社交媒体是多种多样的。

Begel[Beg10] 和他的同事在文章中讲到软件工程中社交媒体的发展和应用：

围绕软件开发的社会化过程……极大程度上取决于工程师的能力——找到有相似目标和互补技能的个人并将他们连接起来，使团队成员的交流和整个团队都表现得更加和谐，让他们在整个软件生命周期中进行合作和协调，并保证他们的产品在市场上能获得成功。

从某种意义上来说，这种“连接”和面对面的交流一样重要。团队规模越大，社交媒体的价值越大，如果团队在地域上是分离的，这种价值就更大了。

首先，社交网络是为软件项目设定的。软件团队可以通过网络从团队成员、利益相关者、技术人员、专家和其他商业人士那里收集经验，这些人已经受邀参与该网络（如果该网络是私有的）或任何兴趣团体（如果该网络是公有的）。而且无论出现什么状况、疑惑或难题，社交网络都可以做到这一点。社交媒体有很多不同的形式，每一种在软件工程工作中都有一席之地。

博客可用来发表一系列短文以描述系统的重要方面，或者用来发表针对尚处于开发中的软件特性或功能的看法。其重要性还可体现在“软件公司经常用博客非常有效地与他们的雇主及客户分享技术信息和观点，内外兼顾。” [Beg10]

引述 内容很重要，对话同样重要。

John Munsell

微博（如 Twitter）由软件工程网络成员使用，对关注他们的人发表短消息。因为这些文章是即时的，而且在各种移动平台上都能阅读，所以信息的分散达到了实时性。软件团队遇到问题时可以随时召开临时会议，出现难题时可以随时寻求专业帮助，也可以实时向利益相关者传达项目的某些方面。

在 targeted on-line 论坛上，参与者可以发布问题、观点、案例或任何其他相关信息。一个技术性的问题在发布之后的几分钟之内就可能得到多个“答复”。

社交网站（如 Facebook、LinkedIn）使软件开发者和相关技术人员达到分离化的连接。同一个社交网站上的“好友”可以找到具有相关应用领域或者要解决问题的知识或经验的好友的好友。以社交网络泛型为基础建立的专业性私有网络可以在组织内部使用。

大多数社交媒体都能组织有相似兴趣的用户形成“社区”。例如，一个由擅长实时嵌入式系统的软件工程师组成的社区，可以为相关领域的个人或团队之间的联系提供有效的方式，以改善他们的工作。随着社区的发展，参与者会讨论技术趋势、应用场景、新型工具和其他软件工程知识。最后，网址收藏夹网站（比如 Delicious、Stumble、CiteULike）为软件工程师或软件团队提供平台，让他们可以为有类似想法的个人组成的社交媒体社区推荐网页资源。

需要着重强调的是，在软件工程工作中使用社交媒体时，不能忽视隐私和安全问题。软件工程师所做的大多数工作可能是他们的雇主专有的，因此将其公开是有害的。所以，一定要在社交媒体的优点和私有信息不受控制的公开之间做出权衡。

6.7 软件工程中云的应用

云计算为我们提供了一种机制,以获取各种软件工作产品、人工制品以及与项目相关的信息。它各处都能运行,并能消除很多软件项目对于设备依赖的限制;可以让软件团队成员进行独立于平台的、低风险的新型软件工具的实验,并提供对这些工具的反馈;可以提供新的分配方法和 β 软件测试;可以提供针对内容和配置管理的更先进的方法(第29章)。

鉴于云计算可以完成上述工作,它很有可能影响软件工程师们组织团队的方式、工作的方法、交流和连接的方式,以及管理软件项目的方式。无论团队成员使用何种平台、处于什么位置,都可以即时获取某个团队成员开发出的软件工程信息。

从根本上说,信息迅速扩散并极度扩展,这也改变了软件工程动态,并对软件工程中人的作用产生了深远的影响。

但是软件工程中的云计算不是没有风险的[The13]。云分布在多个服务器,其构造和服务往往不受软件团队的控制。因此,云有很多缺点,且存在可靠性和安全性风险。云提供的服务越多,相对的开发环境就越复杂。这些服务是否能与其他供应商提供的服务相匹配?这体现了云服务在协同性上的风险。最后,如果云成为开发环境,其服务必须强调可用性和性能。而这些属性有时会与安全性、保密性和可靠性相冲突。

但是从人文的角度来看,与存在的风险相比,云为软件工程师提供了更多的好处。Dana Gardner[Gar09]将其优点总结如下(带有警告):

软件开发中任何涉及社交或合作的地方都很可能会用到云。项目管理、进度安排、任务列表、需求和缺陷管理作为核心团队功能会很好地进行自我调整,其中,沟通对于项目同步以及使团队所有成员——无论他们在哪里——处于同一场景非常重要。当然,需要严重警惕的是——如果你的公司是想设计产品中的嵌入式软件,那么不推荐使用云。想象一下得到苹果公司关于下一版iPhone的项目计划会怎样。

如Gardner所叙述的那样,云的关键优势之一是其增强了“软件开发的社会和协作方面”。在下一节中,你会更多地了解协作工具。

6.8 协作工具

Fillipo Lanubile及其同事[Lan10]认为20个世纪的软件开发环境(SDE)已变成协作开发环境(Collaborative Development Environments, CDE)。^①他们是这样论述的:

工具对于团队成员之间的协作是很有必要的,它们能实现简易化、自动化以及对整个开发过程的控制。全球化软件工程特别需要充足的工具支持,因为距离因素对交流的消极影响直接或间接地加重了协作和控制问题。

在CDE中用到的很多工具和本书第二、三、四部分提到的辅助软件工程活动的工具没有什么不同。但是,有价值的CDE会提供为改善协同工作特别设计的一系列服务[Fok10]。这些服务包括:

- 命名空间使项目团队可以用加强安全性和保密性的方式存储工作产品和其他信息,仅允许有权限的人访问。

引述 他们不再称其为网络,而是称其为云计算。我不纠结名称,随便叫什么都可以。

Larry Ellison

建议 云是软件工程信息的强大知识库,但你必须要考虑第29章所讨论的变更控制问题。

97

^① 协作开发环境(CDE)的概念是由Grady Booch[Boo02]提出的。

98

- 进度表可以协调会议和其他项目事件。
- 模板可以使团队成员在创建工作产品时保持一致的外形和结构。
- 度量支持可以量化每个成员的贡献。
- 交流分析会跟踪整个团队的交流并分离出模式，应用于需要解决的问题或状况。
- 工件收集可以通过回答以下问题的方式组织工作产品和其他项目制品：“是什么导致了某个特定的变更？可以跟哪个讨论过特定制品的人商讨有关变更？（团队）成员的个人工作是如何影响他人的工作的？” [Fok10]

提问 协作开发环境中通用的服务有哪些？

软件工具 协作开发环境

[目标] 随着软件开发不断全球化，软件团队需要的不仅仅是开发工具。他们需要的是一套服务，使得团队成员在本地和远程都能协作。

[机制] 此类工具和服务能使团队建立起协作工作的机制。CDE 可以实现 6.6 节所描述的多种或所有服务，同时也能为实现过程管理（第 4 章）提供本书中讨论的传统软件工程工具。

[代表性工具]^①

- GForge。一种包括项目和代码管理设施的协作环境 (<http://gforge.com/gf/>)。
- OneDesk。为开发者和利益相关者提供创造和管理项目工作空间的协作环境 (www.onedesk.com)。
- Rational Team Concert。一个深度的、协作生命周期管理系统 (<http://www-01.ibm.com/software/rational/products/rtc/>)。

6.9 全球化团队

在软件领域，全球化不仅仅意味着货物和服务的跨国交流。在过去的几十年中，由位于不同国家的软件团队共同建造的主要软件产品越来越多。这些全球化软件开发（GSD）团队具备传统软件团队（6.4 节）所具有的很多特点，但是 GSD 团队还会面临其他特有的挑战，包括协调、合作、交流及专业决策。本章前面已经讨论了协调、合作和交流的方法。对所有软件团队来说，决策问题因以下四个因素而变得复杂 [Gar10]：

- 问题的复杂性。
- 与决策相关的不确定性和风险。
- 结果不确定法则（比如，工作相关的决策会对另外的项目目标产生意外的影响）。
- 对问题的不同看法才是导致不同结论的关键。

对于 GSD 团队，协调、合作和沟通方面的挑战对决策具有深远的影响。图 6-2 解释了 GSD 团队所面临的距离挑战的影响。距离使交流复杂化，但同时也强调对协调的需求。距离也产生了由文化差异导致的障碍和复杂性。障碍和复杂性会减弱交流（比如信噪比的降低）。在这种动态环境中固有的问题会导致项目变得不稳定。

引述 越来越多的公司管理者在应对不同的文化。公司变得全球一体化，但团队却被分开并散布在全球。

Carlos Ghosn,
Nissan

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

99

虽然没有能够彻底改正图 6-2 中各种关系的银弹，但运用有效的 CDE（6.6 节）可有助于减弱距离的影响。

6.10 小结

一个成功的软件工程师必须掌握技术。此外，还必须对自己承担的义务负责、清楚同伴的需求、诚实地评估产品和项目、能适应压力、公平地对待同伴，还要关注细节。

软件工程心理学包括：个人认知和激励，软件团队的群体动力及公司的组织行为。为了改善交流和合作，软件团队成员可以承担跨界角色。

一个成功的（“有凝聚力的”）软件团队比普通团队更多产、更有动力。为实现高效，软件团队必须有目标意识、参与意识、信任意识以及进步意识。此外，必须避免“毒性”——混乱和消极的工作环境、不适合的软件过程、软件团队中模糊的角色定义以及不断暴露的故障。

团队结构有很多种。有些团队是分层结构，另外一些团队则喜欢依赖于个人主动性的松散结构。敏捷团队采用敏捷理论，一般来说比大多数传统软件团队更自主。敏捷团队强调沟通、简单、反馈、勇气和尊重。

社交媒体逐渐成为很多软件项目中必需的部分。博客、微博、论坛和社交网络能使软件工程师社区更有效地进行交流和协作。

云计算对于软件工程师组织团队的方式、工作的方式、交流和连接的方式以及管理软件项目的方式都具有潜在的影响。云能够加强软件开发的社交和协作方面，它的好处远远超过风险。

协作开发环境包含很多服务，可以增强软件团队的交流和协作。这些环境对全球软件开发尤其有益，因为地理上的分离会对软件工程的成功产生障碍。

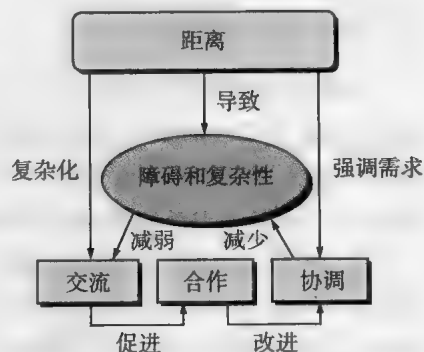


图 6-2 影响 GSD 团队的因素（改自 [Cas06]）

100

习题与思考题

- 6.1 根据你对优秀软件开发者的观察，说出三个共同的个人特质。
- 6.2 怎样能做到“毫无保留地诚实”，同时又不被（其他人）视为有侮辱意图或有攻击性？
- 6.3 软件团队是如何构建“人工边界”来降低与其他人交流的能力的？
- 6.4 编写一个简要场景，描述 6.2 节中的每个“跨界角色”。
- 6.5 在 6.3 节中，我们提到目标意识、参与意识、信任意识和进步意识是高效软件团队的重要属性。在团队成立时，谁应该对建立这些属性负责？
- 6.6 对于团队的四种组织模式（6.4 节）：（1）保险公司的 IT 部门；（2）国防项目承包商的软件工程项目小组；（3）开发电脑游戏的软件小组；（4）大型软件公司。你认为哪种最有效？并阐明理由。
- 6.7 如果要选择敏捷团队区别于传统软件团队的一个属性，你会选哪个？
- 6.8 针对 6.6 节描述的软件工程师工作社交媒体的形式，选出你认为最有效的并说明理由。
- 6.9 编写场景，使 SafeHome 团队成员可以在软件项目中利用一种或多种形式的社交媒体。
- 6.10 近来，云成为计算领域中的一个热门概念。运用有关为改善软件工程师工作而特别设计的服务的具体例子，描述云是如何提升软件工程师组织的价值的。

101

- 6.11 研究一下 6.8 节软件工具介绍中提到的某种 CDE 工具（或导师指定的一种工具），准备在班级中对工具的功能进行简要介绍。
- 6.12 参见图 6-2，距离为什么会使交流复杂化？距离为什么会强调对协调的需求？距离会导致哪些种类的障碍和复杂性？

扩展阅读与信息资源

很多书都讲到了软件工程中人的因素，但有两本书是公认的经典著作。Jerry Weinberg（《The Psychology of Computer Programming》，Silver Anniversary Edition, Dorset House, 1998）首次提到构建计算机软件人员的心理学。Tom DeMarco 和 Tim Lister（《Peopleware: Productive Projects and Teams》，2nd ed., Dorset House, 1999）讨论了软件开发的主要挑战在于人而非技术。

以下书籍提供了针对软件工程中人的有用观察：Mantle 和 Lichty（《Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams》，Addison-Wesley, 2012），Fowler（《The Passionate Programmer》，Pragmatic Bookshelf, 2009），McConnell（《Code Complete》，2nd ed., Microsoft Press, 2004），Brooks（《The Mythical Man-Month》，2nd ed., Addison-Wesley, 1999），Hunt 和 Tomas（《The Pragmatic Programmer》，Addison-Wesley, 1999）。Tomayko 和 Hazzan（《Human Aspects of Software Engineering》，Charles River Media, 2004）以 XP 为重点，讲述了软件工程中的心理学和社会学。

Rasmussen（《The Agile Samurai》，Pragmatic Bookshelf, 2010）和 Davies（《Agile Coaching》，Pragmatic Bookshelf, 2010）论述了敏捷开发中人的因素。有关敏捷团队的重要方面可参见 Adkins 的书（《Coaching Agile Teams》，Addison-Wesley, 2010），还有 Derby、Larsen 和 Schwaber 的书（《Agile Retrospectives: Making Good Teams Great》，Pragmatic Bookshelf, 2006）。

解决问题是一种独特的人类活动，这方面的著作有：Adair（《Decision Making and Problem Solving Strategies》，Kogan Page, 2010），Roam（《Unfolding the Napkin》，Portfolio Trade, 2009），Wananabe（《Problem Solving 101》，Portfolio Hardcover, 2009）。

Tabaka（《Collaboration Explained》，Addison-Wesley, 2006）提供了如何在软件团队中进行合作的指导。Rosen（《The Culture of Collaboration》，Red Ape Publishing, 2009）、Hansen（《Collaboration》，Harvard Business School Press, 2009）和 Sawyer（《Group Genius: The Creative Power of Collaboration》，Basic Books, 2007）提供了改善技术团队合作的策略和实践性指导。

以培养人员创新性为主题的书有：Gray、Brown 和 Macanufo（《Game Storming》，O'Reilly Media, 2010），Duggan（《Strategic Intuition》，Columbia University Press, 2007）和 Hohmann（《Innovation Games》，Addison-Wesley, 2006）。

Ebert（《Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing》，Wiley-IEEE Computer Society Press, 2011）描述了全球软件开发的整体情况。Mite 及其同事（《Agility Across Time and Space: Implementing Agile Methods in Global Software Projects》，Springer, 2010）编写了有关全球开发中使用敏捷团队的论文集。

网上有讨论软件工程中人员方面的大量信息资源，软件过程相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

建 模

在本书这一部分，读者将学到构建高质量需求模型和设计模型的基本原则、概念和方法。接下来的章节将涉及如下问题：

- 指导软件工程实践的概念和原则有哪些？
- 什么是需求工程？什么是能导致良好需求分析的基本概念？
- 如何创建需求模型？它包含哪些元素？
- 好的设计包含哪些元素？
- 体系结构设计如何为其他的设计活动建立架构，使用什么模型？
- 如何设计高质量软件部件？
- 在设计用户接口时使用什么概念、模型和方法？
- 什么是基于模式的设计？
- 使用什么特别的策略和方法来设计 WebApp？
- 使用什么特别的策略和方法来设计移动 App？

一旦解决了以上问题，准备工作也就完成了，可以马上开始软件工程应用实践了。

指导实践的原则

要点浏览

概念：软件工程实践是软件计划和开发时需要考虑的方方面面，包括概念、原则、方法和工具等。指导实践的原则成为软件工程实施的基础。

人员：软件工程实践由实践者（软件工程师）和软件项目经理共同完成多种软件工程任务。

重要性：软件过程为每个开发计算机系统或产品的人提供了成功抵达目的地的路线图。实践为你提供了沿路驾驶的细节，它会告诉你哪里有桥、哪里有路障、哪里有岔路，它帮助你理解一些必须理解的并且必须遵循的快速安全驾驶概念和原则，它指示你如何驾驶、在哪里应该减速、在哪里应该加速。在软件工程中，

实践就是把软件由想法转化为现实时你天天应该做的事情。

步骤：不管选择哪种过程模型，都必须运用实践三要素——概念、原则和方法。实践的第四个要素是工具，工具为方法的应用提供支持。

工作产品：实践贯穿于整个技术活动中。这些技术活动开发出由所选软件过程模型定义的所有工作产品。

质量保证措施：首先，要深刻理解目前工作所用到的原则（例如设计）。然后，确信你已经选定了一种合适的方法；确信你已经理解了如何运用这种方法以及使用适合此任务的自动工具，并且坚信需要一些技术来保证工作产品的质量。

在一本探讨软件工程师生活和思想的书中，Ellen Ullman[Ull97]通过如下生活片段描述了重重压力下软件工作者的思索：

我对于时间已经没有概念。在这间办公室里，没有窗户也没有时钟，只有红色 LED（发光二极管）微波显示器在闪烁，它不断地闪现着 12:00，12:00，12:00，12:00。Joel 和我已经折腾了好几天，有一个 bug，一个很难处理的 bug。这红色的闪烁脉冲就像我们的大脑，一直在以相同的速率闪动着……

我们在做什么？……具体是什么我现在也不知道。我们可能是在帮助可怜的病人，或者是在分布式数据库协议上调整一组底层例程来验证比特流——这些我并不关心。我应该关心，或者说另一个我应该关心——在此之后，或许是我们从这间到处都是电脑的房间中出来的时候，我会非常关心为什么、为了谁以及为了什么目标而开发软件。但是现在不。我已经穿过了一层隔膜，在这里真实世界及其用处都已不再重要，我是软件工程师……

这段文字展现了软件工程实践的黑暗画面，而这种情况很可能和本书大多数读者都有关系。

关键概念

- 编码原则
- 沟通原则
- 核心原则
- 部署原则
- 设计建模原则
- 生存建模原则
- 建模原则
- 策划原则
- 实践
- 过程
- 需求建模原则
- 测试原则

计算机软件开发人员日常从事的艺术、工艺或者规范性活动^①就是软件工程。那什么是软件工程“实践”？一般来讲，实践就是软件工程师每天使用的概念、原则、方法和开发工具的集合。实践使得项目经理可以管理软件项目，保证软件工程师开发计算机程序。实践利用由必要技术和管理组成的软件过程模型，保证开发工作顺利开展。实践将一些杂乱的、容易被忽视的方法转化为更具组织性、更高效并且更容易获得成功的重要东西。

软件工程实践的各个不同方面将在本书的剩余部分介绍。本章的重点是指导软件工程实践通用的原则和概念。

7.1 软件工程知识

Steve McConnell[McC99] 在编辑出版的《IEEE Software》中发表了以下评论：

许多软件实践者认为软件工程知识几乎无异于特定技术知识，比如 Java、Perl、html、C++、Linux、Windows NT 等。这些特定技术知识是用来进行计算机程序设计的。如果某人派你去编写一个 C++ 程序，你就得掌握关于 C++ 的知识来完成编程工作。

经常听到人们说，软件开发知识的半衰期为三年：现在你需要知道的那些知识，在三年内有一半将过时。在技术相关的知识领域内，这种说法可能是正确的。但还有另一种软件开发知识——我称其为“软件工程原则”——并没有三年半衰期的说法。这些软件工程原则可以为专业程序设计人员在其整个职业生涯内提供服务。

McConnell 继续论述了软件工程知识体（大约在 2000 年）已经演变为“稳定的核心”，他估计该知识体大约提供了“开发一个复杂系统所需的 75% 知识”。但是这个“稳定的核心”里面都有些什么呢？

这些年来，我们看到了 iOS 和安卓系统这样的创新性操作系统，以及 Java、Python 和 C# 这样的创新性语言。但是，如 McConnell 所述，核心原则——指导软件工程师工作的基本思想——依然能够提供从软件工程模型、方法和工具中得来的可以应用和评价的基本原理。

[105]

7.2 核心原则

软件工程是以一系列核心原则作指导的，这些核心原则为应用具有重大意义的软件过程以及执行有效的软件工程方法提供了帮助。在过程级，核心原则建立了哲学基础，从而指导软件开发团队执行框架活动和普适性活动、引导过程流以及生产一系列软件工程产品。在实践级，核心原则建立了一系列价值和规则，为分析问题、设计解决方案、实现和测试解决方案以及最终在用户社区部署软件提供指导。

引述 理论上，理论与实践没有区别，而事实上，差别是存在的。

Jan van de
Snepscheut

第 2 章曾提出一系列跨越软件工程过程和实践的通用原则：（1）为最终用户提供价值；（2）保持简洁；（3）保持愿景（产品和计划）；（4）认识（必须理解）到是别人在消费你生产的产品；（5）面向未来；（6）提前计划复用；（7）认真思考！虽然这些通用原则非常重要，但是以这样高度抽象的语言来表述，使得它们有时很难转化为日常的软件工程实践。以下将更为具体地着眼于指导过程和实践的核心原则。

① 一些作者主张使用这些词语中的一个，而排除其他词语。事实上，软件工程包括所有这三个含义。

7.2.1 指导过程的原则

本书的第一部分讨论了软件过程的重要性，描述了提供给软件工程的许多不同的过程模型。但无论模型是线性的或迭代的，是传统的或敏捷的，都可以用适用于所有过程模型的普通过程框架来描述。以下这组核心原则适用于框架，并可延伸至每一个软件过程。

原则 1：敏捷。关于你所选择的过程模型是传统的还是敏捷的，敏捷开发的基本原则会提供判断方法。你所做的工作的每一方面都应着重于活动的经济性——保持你的技术方法尽可能简单，保持你的工作产品尽可能简洁，无论何时尽可能根据具体情况做出决定。

106

原则 2：每一步都关注质量。每个过程活动、动作及任务的出口条件都应关注所生产的工作产品的质量。

原则 3：做好适应的准备。过程不是信奉经验，其中没有信条。必要的时候，就让你的方法适应于由问题、人员以及项目本身施加的限制。

原则 4：建立一个有效的团队。软件工程过程和实践是重要的，但最根本的还是人。必须建立一个彼此信任和尊重的自组织团队。^①

原则 5：建立沟通和协调机制。项目失败是由于遗漏了重要信息，或是利益相关者未能尽力去创造一个成功的最终产品。这些属于管理的问题，必须设法解决。

原则 6：管理变更。管理变更的方法可以是正式的或非正式的，但是必须建立一种机制，来管理变更要求的提出、变更的评估、变更的批准以及变更实施的方式。

原则 7：评估风险。在进行软件开发时会出现很多问题。建立应急计划是非常重要的。某些应急计划会成为安全工程任务的基准（第 27 章）。

原则 8：创造能给别人带来价值的工作产品。唯有那些能为其他过程活动、动作或任务提供价值的工作产品才值得创造。每一个工作产品都会作为软件工程实践的一部分传递给别人。需求功能和特性表会传递给设计开发人员，设计会传递给编码人员，等等。一定要确保工作产品所传达的是必要信息，不会模棱两可或缺不全。

本书第四部分的重点是项目管理和过程管理问题，以及这些原则各个方面的细节。

7.2.2 指导实践的原则

软件工程实践有一个最重要的目标——按时交付包含了满足所有利益相关者要求的功能和特性的高质量、可运行软件。为了实现这一目标，必须采用一系列的核心原则来指导技术工作。这些原则的优点是不用考虑你所使用的分析方法、设计方法和构建技术（例如，程序设计语言、自动工具），也不用考虑你所选用的验证和确认方法。以下列举的一组核心原则是软件工程实践的基础。

107

原则 1：分治策略（分割和攻克）。更具技术性的表达方式是：分析和设计中应经常强调关注点分离（Separation of Concern, SoC）。一个大问题分解为一组小元素（或关注点）之后就比较容易求解。从概念上讲，每个关注点都传达了可以开发且在某些情况下可被确认的特定功能，这与其他关注点是无关的。

原则 2：理解抽象的使用。在这一核心原则中，抽象就是对系统中一些复杂元素的简单

建议 每一个计划和每一个团队都是独立的。这就意味着，必须使过程尽可能地适应于需求。

引述 事实是你总是知道该做的正确事情。困难的是去做。

General
H. Norman
Schwarzkopf

① 有效的软件团队的特点已在第 6 章讨论过。

化,用一个专有用语来交流信息。我使用报表这一抽象,是假设你可以理解什么是报表,报表表示的是目录的普通结构,可以将经典的功能应用其中。在软件工程实践中可以使用许多不同层次的抽象,每个抽象都通告或暗示着必须交流的信息。在分析和设计中,软件团队通常从高度抽象的模型开始(如报表),然后逐渐将这些模型提炼成较低层次的抽象(如专栏或SUM功能)。

Joel Spolsky[Spo02]提出,“在某种程度上,即使是正规的抽象也都是有漏洞的”。抽象的意图是减弱交流细节的需求。但有时,这些省略的细节积少成多,将会导致问题层层“渗漏”。由于对细节不了解,所以对问题产生的原因也不容易分析。

原则3:力求一致性。无论是创建需求模型、开发软件设计、开发源代码还是创建测试用例,一致性原则都建议采用用户熟悉的上下文以使软件易于使用。例如,为WebApp设计一个用户界面,一致的菜单选择、一致的色彩设计以及一致的可识别图标都有助于使界面在人体工程学方面更为合理。

原则4:关注信息传送。软件所涉及的信息传送是多方面的——从数据库到最终用户、从遗留的应用系统到WebApp、从最终用户到图形用户界面(GUI)、从操作系统到应用问题、从一个软件构件到另一个构件,这个列表几乎是无穷无尽的。在每一种情况下,信息都会流经界面,因而,就有可能出现错误、遗漏或者歧义的情况。这一原则的含义是必须特别注意界面的分析、设计、构建以及测试。

原则5:构建能展示有效模块化的软件。对重要事务的分割(原则1)建立了软件的哲学,模块化则提供了认知这一哲学的机制。任何一个复杂的系统都可以被分割成许多模块(构件),但是好的软件工程实践不仅如此,它还要求模块必须是有效的。也就是说,每个模块都应该专门集中表示系统中约束良好的一个方面——其功能具有内聚性或局限于所表示的内容范围。另外,模块应当以相对简单的方式关联起来——每个模块与其他模块、数据源以及环境方面都应是低耦合的。

原则6:寻找模式。Brad Appleton[App00]指出了如下观点。

在软件界,模式的目标是建立一个典集,帮助软件开发者解决整个软件开发过程中反复出现的问题。模式还有助于创建一种共同语言,交流有关这些问题及其解决办法的见解和经验。将这些解决办法以及它们之间的关系正式编纂成典可以使我们成功地捕获知识体系,这一知识体系中明确了对满足用户需求的良好体系结构的认识。

设计模式广泛应用于系统工程和系统整合问题,允许复杂系统中的组件独立发展。

原则7:在可能的时候,用大量不同的观点描述问题及其解决方法。当我们用大量不同的观点检测一个问题及其求解方法时,就很有可能获得更深刻的认识,并发现错误和遗漏。例如,需求模型可以用面向场景的观点、面向类的观点或面向行为的观点(第9章和第11章)来陈述,每个观点都提供了一个对问题及其需求的不同看法。

原则8:记住,有人将要对软件进行维护。从长期看,缺陷暴露出来时,软件需要修正;环境发生变化时,软件需要适应;利益相关者需要更多功能时,软件需要增强。如果可靠的软件工程实践能够贯穿于整个软件过程,就会便于这些维护活动的实施。

虽然这些原则不能包含构建高质量软件所需要的全部内容,但是它们为本书讨论的每种

建议 未来的软件工程师使用模式(第16章)来获取知识和经验。

建议 利用多个不同观点检测同一个问题,这样能避免狭隘的视野。

软件工程方法奠定了基础。

7.3 指导每个框架活动的原则

109

以下几节关注的是对作为软件过程部分的每一个通用框架活动的成功产生重要影响的原则。在很多情况下，所讨论的每个框架活动的原则都是对 7.2 节中提出的原则的提炼。它们只是处于较低抽象层次的核心原则。

7.3.1 沟通原则

在分析、建模或规格说明之前，客户的需求必须通过沟通活动来收集。有一些客户问题可能适合于使用计算机求解，这时软件人员就要对客户请求做出响应。沟通开始了，但是从沟通到理解这条路并不平坦。

高效的沟通（与其他技术人员的沟通、与客户和其他利益相关者的沟通、与项目经理的沟通）是一个软件工程师所面临的最具挑战性的工作。在这里，我们将讨论与客户沟通的原则。不过，许多原则同样适用于软件项目内部的沟通。

原则 1：倾听。一定要仔细倾听讲话者的每一句话，而不是急于叙述你对这些话的看法。如果有什么事情不清楚，可以要求他澄清，但是不要经常打断别人的讲述。当别人正在陈述的时候不要在言语或动作上表现出异议（比如转动眼睛或者摇头）。

建议 在沟通之前确定你理解了其他人的观点，知道他需要什么，然后倾听。

原则 2：有准备的沟通。在与其他人碰面之前花点时间去理解问题。如果必要的话，做一些调查来理解业务领域的术语。如果你负责主持一个会议，那么在开会之前准备一个议事日程。

引述 简洁的问题和简洁的回答是解决问题最好的办法。

Mark Twain

原则 3：沟通活动需要有人推动。每个沟通会议都应该有一个主持人（推动者），其作用是：（1）保持会议向着有效的方向进行；（2）调解会议中发生的冲突；（3）确保遵循我们所说的沟通原则。

原则 4：最好当面沟通。但是，如果能把一些相关信息写出来，通常可以工作得更好，例如可以在集中讨论中使用草图或文档草稿。

原则 5：记笔记并且记录所有决定。任何解决方法都可能存在缺陷。参与交流的记录员应该记录下所有要点和决定。

原则 6：保持通力协作。当项目组成员的想法需要汇集在一起用以阐述一个产品或者某个系统的功能或特性时，就产生了协作与协调的问题。每次小型的协作都可能建立起项目成员间的相互信任，并且为项目组创建一致的目标。

110

原则 7：把讨论集中在限定的范围内。在任何交流中，参与的人越多，话题转移到其他地方的可能性就越大。最简便的方法就是保持谈话模块化，只有某个话题完全解决之后才能开始别的话题（不过还应该注意原则 9）。

原则 8：如果某些东西很难表述清楚，就采用图形表示。语言沟通的效果很有限，当语言无法表述某项工作的时候，草图或者绘图通常可以让表述变得更为清晰。

原则 9：（1）一旦认可某件事情，转换话题；（2）如果不认可某件事情，转换话题；（3）如果某项特性、功能不清晰或当时无法澄清，转换话题。交流如同所有其他软件工程活动一样需要时间，与其永无止境地迭代，不如让参与者认识到还有很多话题需要讨论（参见原则 2），“转换话

提问 如果无法与客户就项目相关问题达成一致，将会发生什么？

题”有时是达到敏捷交流的最好方式。

原则 10：协商不是一场竞赛或者一场游戏，双赢才能发挥协商的最大价值。很多时候软件工程师和利益相关者必须商讨一些问题，如功能和特性、优先级和交付日期等。若要团队合作得好，那么各方要有一个共同的目标，并且协商还需要各方的协调。

信息栏 客户与最终用户的差别

软件工程师会与许多利益相关者交流，但是客户与最终用户对将采用的技术影响最大。有的时候客户与最终用户是相同的人，但是对于许多项目来说，客户与最终用户是不同的人，他们为不同商业组织的不同管理者工作。

客户是这样的人（或组织）：（1）最初要求构建软件的人；（2）为软件定义全部业务目标的人；（3）提供基本的产品需求

的人；（4）为项目协调资金的人。在产品业务或者系统业务中，客户通常是销售部门；在 IT 环境下，客户或许是一个业务单位或部门。

最终用户是这样的人（或组织）：（1）为了达到某种商业目的而将真正使用所编写软件的人；（2）为达到其商业目的将定义软件可操作细节的人。

SafeHome 沟通问题

[场景] 软件工程开发团队工作场所。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员。

[对话]

Ed：对这个 SafeHome 的项目你们听到什么了？

Vinod：第一次会议安排在下周。

Jamie：我已经做了一些调查，但是进行得不那么顺利。

Ed：你的意思是？

Jamie：是这样，我给 Lisa Perez 打了一个电话，她是销售部经理。

Vinod：然后呢……

Jamie：我想让她告诉我关于 SafeHome 的特性和功能……之类的事情。可是，她开始问我一些关于安全系统、监视系统等方面的问题，这些我并不精通。

Vinod：这对你有何启示？

（Jamie 耸了耸肩）

Vinod：那个销售部门需要我们扮演顾问的角色，我们最好在第一次会议之前对这个产品领域做一些了解。Doug 说过他希望我们与客户“协作”，这样才能更好地了解如何开发。

Ed：也许你应该去她的办公室，电话不适合做这样的工作。

Jamie：你们说的都对。我们应该努力做好这方面的工作，做好早期交流。

Vinod：我看到 Doug 在看“需求工程”方面的书，我敢打赌这本书上肯定罗列了一些好的交流原则，我要借来看看。

Jamie：好主意，然后你就可以教我们啦。

Vinod（微笑）：没问题。

111

7.3.2 策划原则

沟通活动可以协助软件开发团队定义其全局目标（当然，主题会随着时间的推移而变

化)。可是,理解这些目标与为达到这些目标而制定计划并不是一回事。策划活动包括一系列管理和技术实践,可以为软件开发团队定义一个便于他们向着战略目标和战术目标前进的路线图。

不管我们做多少努力,都不可能准确预测软件项目会如何进展。也不存在简单的方法来解决各类可能出现的问题:确定不可预见的技术问题,在项目后期还有什么重要信息没有掌握,以及会出现什么误解,或者会有什么商务问题发生变化。然而,软件团队还是必须制定计划。

有很多不同的制定计划的哲学。^①一些人是“最低要求者”,他们认为变化常常会消除详细计划的必要性;另一些人是“传统主义者”,他们认为计划提供了有效的路线图,并且计划得越详细,团队损失的可能性就越小;也有一些人是“敏捷主义者”,他们认为快速制定计划是必需的,而路线图将会在真正的软件开发工作开始时浮现出来。

要做什么?在许多项目中,“过度计划”是浪费时间并且是在做无功(因为事物有太多的变化),但是“最起码的计划”是制止混乱的良方。就像在生活中的很多事情一样,适度执行计划足以为团队提供有用的指导——不多也不少。无论多么严格地制定计划,都应该遵循以下原则。

原则 1: 理解项目范围。如果你不知道要去哪里,就不可能使用路线图。范围可以为软件开发团队提供一个目的地。

原则 2: 让利益相关者参与策划。利益相关者能够限定一些优先次序,确定项目的约束。为了适应这种情况,软件工程师必须经常商谈交付的顺序、时间表以及其他与项目相关的问题。

原则 3: 要认识到计划的制定应按照迭代方式进行。项目计划不可能一成不变。在工作开始的时候,有很多事情有可能改变,那么就必须调整计划以适应这些变化。另外,迭代式增量过程模型指出了要根据用户反馈的信息(在每个软件增量交付之后)重新制定计划。

原则 4: 基于已知的估算。估算的目的是基于项目组对将要完成工作的当前理解,提供一种关于工作量、成本和任务工期的指标。如果信息是含糊的或者不可靠的,估算也将是不可靠的。

原则 5: 计划时考虑风险。如果团队已经明确了哪些风险最容易发生且影响最大,那么应急计划就是必需的了。另外,项目计划(包括进度计划)应该可以调整,以适应那些可能发生的一种或多种风险。要考虑由于项目资产丢失或损坏而暴露的问题。

原则 6: 保证可实现性。人们不能每天百分百地投入工作。噪音总能侵入人们的交流之中。现实生活常常会有疏忽与含糊。变化总是在发生。甚至最好的软件工程师都会犯错误,这些现实情况都应该在项目制定计划的时候考虑。

原则 7: 调整计划粒度。粒度主要指项目计划细节的精细程度。“细粒度”的计划可以提供重要的工作任务细节,这些细节是在相对短的时间段内计划完成的(这样就常常会有跟踪和控制的问题)。“粗粒度”的计划提供了更宽泛的长时间工作任务。通常,粒度随项目的进行而从细到粗。对于几个星期或少数几个月时间里的活动,我们可以详细地策划项目,而在

引述 在为战役做准备时,我经常发现,计划是无用的,但做计划是必不可少的。

General Dwight
D. Eisenhower

网络资源 关于计划和项目管理信息的优秀资料库见 www.4pm.com/。

引述 成功更多的是一种协调能力,而不是天赋。

An Wang

关键点 粒度表明项目计划中表述和控制的元素的详细程度。

^① 关于软件项目制定计划和管理的具体讨论在本书第四部分给出。

很多个月内都不会发生的活动则不需要细化（太多的东西将会发生变化）。

113

原则 8：制定计划以确保质量。计划中应该确定软件开发团队如何确保开发的质量。如果要执行正式技术评审^①的话，应该将其列入进度；如果在构建过程中用到了结对编程（第5章），那么在计划中要明确描述。

原则 9：描述如何适应变化。即使最好的策划也有可能被无法控制的变化破坏。软件开发团队应该确定在软件开发过程中如何适应变化，例如，客户会随时提出变更吗？如果提出了一个变更，团队是不是要立即实现？变更会带来怎样的影响和开销？

原则 10：经常跟踪并根据需要调整计划。项目每次会落后进度一天的时间。因此，需要每天都追踪计划的进展，找出计划与实际执行不一致的问题所在，当任务进行出现延误时，计划也要随之做出调整。

最高效的方法是软件开发项目组所有成员都参与到策划活动中来，只有这样，项目组成员才能很好地认可所制定的计划。

7.3.3 建模原则

我们可以通过创建模型来更好地理解需要构建的实体。当实体是物理实物（例如一栋建筑、一架飞机、一台机器）时，我们可以构建在形式和形状上都和实物相同只是比实物缩小了的模型。可是，当实体是软件时，我们的模型就是另外一种形式了。它必须能够表现出软件所转换的信息、使转换发生的架构和功能、用户要求的特性以及转换发生时系统的行为。模型必须能在不同的抽象层次下完成那些目标——首先从客户的角度描述软件，然后在更侧重于技术的方面表述软件。

114

在软件工程中要创建两类模型：需求模型和设计模型。需求模型（也称为分析模型）通过在以下三个不同域描述软件来表达客户的需求：信息域、功能域和行为域。设计模型表述了可以帮助开发者高效开发软件的特征：架构、用户界面以及构件细节。

关键点 分析模型表达了客户的需求。设计模型为软件的构造提供了详细的描述。

Scott Ambler 和 Ron Jeffries[Amb02b] 在他们关于敏捷建模的书中定义了一系列建模原则^②，提供给使用敏捷模型（第5章）的人，但也适用于执行建模活动和任务的软件工程师。

原则 1：软件团队的主要目标是构建软件而不是创建模型。敏捷的意义是尽可能快地将软件提供给用户。可以达到这个目标的模型是值得软件团队构建的，但是，我们需要避免那些降低了开发过程的速度以及不能提供新的见解的模型。

原则 2：轻装前进——不要创建任何不需要的模型。每次发生变化时，创建的模型必须是最新的。更重要的是，每创建一个新模型所花费的时间，还不如花费在构建软件上（编码或测试）。因此，只创建那些可以使软件的构建更加简便和快速的模型。

原则 3：尽量创建能描述问题和软件的最简单模型。不要建造过于复杂的软件[Amb02b]。保持模型简单，产生的软件必然也会简单。最终的结果是，软件易于集成、易于测试且易于维护（对于变更）。另外，简单的模型易于开发团队成员理解和评判，从而使持续不断的反馈可以对最终结果进行优化。

① 技术评审在第20章讨论。

② 本节给出的原则因本书目标已经做了删节和改写。

原则 4：用能适应变化的方式构建模型。假设模型将要发生变化，但做这种假设并不草率。例如，由于需求发生变化，就需要迅速改变需求模型。为什么？因为不管怎样它们都会发生变化。所带来的问题是，如果没有相当完整的需求模型，那么所创建的设计（设计模型）会常常丢失重要功能和特性。

原则 5：明确描述创建每一个模型的目的。每次创建模型时，都问一下自己为什么这么做。如果不能为模型的存在提供可靠和正当的理由，就不要再在这个模型上花费时间。

原则 6：调整模型来适应待开发系统。有时需要使模型的表达方式或规则适用于应用问题。例如，一个视频游戏应用需要的建模技术与实时嵌入式控制汽车引擎的软件所需的建模技术或许会完全不同。

原则 7：尽量构建有用的模型而不是完美的模型。当构建需求模型和设计模型时，软件工程师要达到减少返工的目的。也就是说，努力使模型绝对完美和内部一致的做法是不值当的。这是在建议模型应当草率或低质量吗？答案是“不”。但是，对当前建模工作的管理要始终考虑到软件工程的下一步骤的实施。无休止地使模型“完美”并不能满足敏捷的要求。

原则 8：对于模型的构造方法不要过于死板。如果模型能够成功地传递信息，那么表述形式是次要的。虽然软件团队的每个人在建模期间都应使用一致的表达方式，但模型最重要的特性是交流信息，以便软件工程执行下一个任务。如果模型可以成功地做到这一点，不正确的表达方式就可以忽略。

原则 9：如果直觉告诉你模型不太妥当，尽管书面上很正确，那么你也要仔细注意了。如果你是个有经验的软件工程师，就应相信直觉。软件工作中有许多教训——其中有些是潜意识的。如果有些事情告诉你设计的模型注定会失败（尽管你不能明确地证明），你就有理由再花一些时间来检查模型或开发另一个模型。

原则 10：尽可能快地获得反馈。每个模型都应经过软件团队的评审。评审的目的是提供反馈，用于纠正模型中的错误、改变误解，并增加不经意遗漏的功能和特性。

需求建模原则。在过去的 30 年里，人们已经开发出了大量的需求建模方法。研究人员已经弄清了需求分析中的问题及其出现原因，也开发出了各式各样的建模表达方法以及相关启发性解决方法。每一种分析方法都有其独立的观点。不过，所有的分析方法都具有共同的操作原则。

原则 1：必须描述并理解问题的信息域。信息域包括流入系统的数据（来自最终用户、其他系统或者外部设备）、流出系统的数据（通过用户接口、网络接口、报告、图形以及其他方式）以及那些收集和组织的永久性数据对象的数据存储（例如永久存储的数据）。

原则 2：必须确定软件所要实现的功能。软件功能直接为最终用户服务并且为用户可见的特性提供内部支持。一些功能对流入系统的数据进行转换。在其他情况下，有些功能在某种程度上影响着对软件内部过程或外部系统元素的控制。功能可以以不同的抽象层次描述，这些抽象层次从对目标的笼统陈述到对那些必不可少的过程细节的描述。

原则 3：必须描述软件的行为（作为外部事件的结果）。软件的行为受到与外部环境交互的驱动。最终用户提供的输入、由外部系统提供的控制

建议 任一模型的目标都是传递信息。为了实现这一目标，需使用一致的格式。假设你不能解释此模型，那么它始终都只是个模型而已。

引述 工程师在设计时的首要问题是发现真正的问题。

作者不详

关键点 分析模型集中于软件三个属性：要处理的信息，要发布的功能和要展现的行为。

数据或者通过网络收集的监控数据都会引起软件的特定行为。

原则4：描述信息、功能和行为的模型必须以一种能揭示分层（或者分级）细节的方式分解开来。需求建模是解决软件工程问题的第一步。它能使开发者更好地理解问题并且为确定解决方案（设计）准备条件。复杂的问题很难完全解决，基于这样的原因，我们使用“分而治之”的战略。把大的复杂问题划分成很多易于理解的子问题，这就是分解的概念，也是分析建模的关键策略。

原则5：分析任务应该从本质信息转向实现细节。需求建模是以从最终用户角度描述问题为开始的。在没有考虑解决方案的前提下描述问题的“本质”。例如，一个视频游戏需要玩家在他所扮演角色进入一个危险的迷宫时控制其角色行动的方向，这就是问题的本质。实现细节（通常作为设计模型的一部分来描述）指出问题的本质将如何实现，对于视频游戏来说，可能需要用到语音输入，或者键入键盘命令，或者朝某个特定方向移动操纵杆（或者鼠标），或者在空中挥舞动作感应设备。

通过应用这些原则，软件工程师可以系统地解决问题。但这些原则如何应用到实践中呢？对这个问题的回答在第8章到第11章可以找到。

设计建模原则。软件设计模型类似于建筑师的房屋设计方案。首先表达所有需要建造的东西（例如房屋的三维透视图），然后逐渐进行细化，为构建各个细节（例如管线分布）提供指导。类似的，软件设计模型为系统提供了各式各样的不同视图。

现在已经有许多方法可以导出各种软件设计的要素。一部分方法是数据驱动的，它通过数据结构来得到程序构架和最终的处理构件；另一部分是模式驱动的，也就是使用问题域（需求模型）信息来开发架构风格和处理模式；还有一些方法是面向对象的，使用问题域对象来驱动创建数据结构以及操作这些数据结构的方法。不过，无论使用什么方法，都使用同一套设计原则。

原则1：设计可追溯到需求模型。需求模型描述了问题的信息域、用户可见的功能、系统的行为以及一套需求类，需求类把业务对象和为这些对象服务的方法结合在一起。设计模型将这些信息转化为系统架构、一套实现主要功能的子系统以及一套实现需求类的构件。设计模型的各个元素都应该能追溯到需求模型。

原则2：要始终关注待建系统的架构。软件架构（第13章）是待建系统的骨架，它决定着系统接口、数据结构、程序控制流和行为、测试方法以及在建系统的可维护性等。基于上述原因，设计应该从考虑架构开始，在架构确定以后才开始考虑构件级设计。

原则3：数据设计与功能设计同等重要。数据设计是架构设计的基本要素。在设计中数据对象的实现方法绝不能忽略，一个结构良好的数据设计可以简化程序流程，让软件构件设计与实现变得更简单，使得整个处理过程更为高效。

原则4：必须精心设计接口（包括内部接口和外部接口）。系统中构件之间数据流的流动方式大大影响着系统的处理效率、误差传播以及设计简单化等方面。好的接口设计可以让构件集成变得更为容易并能辅助测试人员确认构件的功能。

原则5：用户界面设计必须符合最终用户要求。在任何情况下，界面的设计都应强调使

引述 首先要理解设计是考虑周全而且合理的：一经证明是无误的，就要坚定地追求它；不要因为有人排斥已经确定的意图而动摇取得最终结果的决心。

William
Shakespeare

网络资源 关于设计过程的深入评论及相关设计美学讨论可以访问 http://www.gobookee.net/sea_rch.php?q=aabyan+design+aesthetics。

118 用的方便性。用户接口是软件中可见的部分，无论系统的内部功能多么复杂，无论数据结构多么容易理解，无论系统架构设计有多好，不好的界面设计都会令人感到整个软件很糟糕。

原则 6：构件级设计应是功能独立的。功能上独立是软件构件“单一思想”的度量方法。构件提供的功能应该是内聚的——也就是说，它应该关注于一个且仅仅一个功能或子功能。^①

原则 7：构件之间以及构件与外部环境之间松散耦合。耦合可以通过很多方式来实现，如构件接口、消息传递及全局数据。随着耦合程度的提高，错误传播的概率也会随之提高，整个软件的可维护性也会降低。因此，应该合理地保持较低的构件耦合度。

原则 8：设计表述（模型）应该做到尽可能易于理解。设计的目的是向编码人员、测试人员以及未来的维护人员传递信息，如果设计过于复杂且难以理解，就无法成为一种高效的沟通媒介。

原则 9：设计应该迭代式进行。每一次迭代时，设计者都应该力求简洁。就像大多数创造性活动一样，设计是迭代完成的，第一次迭代的工作是细化设计并纠正错误，之后的迭代就应该让设计变得尽量简单。

原则 10：创建设计模型不包括在敏捷开发的方法中。某些敏捷开发（第 5 章）的倡导者坚持认为代码是唯一必需的设计文档，而设计模型的目标是帮助那些必须维护和开发的系统。那些系统有非常难于理解的内容，既有较高级别目标的代码段，也有在多线程实时环境中与其他模块交互的代码段。

虽然内联代码的文档非常有用，但是通常很难保持代码和代码描述的一致性。设计模型为我们提供了便利，因为生成的抽象层去除了不必要的技术细节，使应用内容和需求结合得更加紧密。

119 补充设计信息可以阐述设计理念，包含对已拒绝的架构设计替代方案的描述。这些信息可以帮助我们看透代码森林。另外，当需要细粒度设计时，这也有助于维护连贯性、一致性。这种架构说明书还有助于各个系统的利益相关者与设计团队及其他团队的沟通。

除了用于能够很快建立原型并实施的小型系统以外，仅用源代码做高级设计是不明智的。敏捷设计文档紧随设计和开发工作。为了避免浪费，对这些文档的阐述工作必须与设计的稳定性相称。在设计的前期阶段，描述文档必须适于各个利益相关者之间的沟通，描述内容越深入则设计越稳定。一种方法是使用建模工具，其生成的可执行模型能对常规敏捷行为进行评估。

在适当应用这些设计原则时，我们的设计会同时展示出内部和外部的质量因素 [Mye78]。外部质量因素是软件属性，即用户所要观察的属性（例如速度、可靠性、更正率、使用率）。内部质量因素是以软件工程师为重点的。他们从技术层面进行高质量的设计。为满足内部质量因素，设计者必须理解基本的设计理念（第 12 章）。

生存建模原则。Breu[Breu10] 描述了生存模型，这一范型中涉及基于模型的开发^②，并与管理和操作面向服务的系统^③相结合。生存模型支持所有利益相关者之间的合作，并提供合适的基于其他模型的抽象化，用于描述系统元素间的相互独立性。以下 8 条原则是建立生

引述 差异并不小，就像 Salieri 和 Mozart 之间的差异。一项又一项的研究表明最优秀的设计师构建的结构更快、更小、更简单、更清晰而且不甚费力。

Frederick P.
Brooks

① 内聚的附加讨论见第 12 章。

② 基于模型的开发（也称为模型驱动工程）建立域模型，即描述一个应用域的特定部分。

③ 面向服务的系统在服务前通过网络基础设施对软件按功能进行打包。

存模型环境的关键。

原则 1：利益相关者为中心的模型应以特定的利益相关者和他们的任务为目标。这意味着允许利益相关者在适当的抽象层级进行操作，其他内容隐藏在较低的层级中。例如，CIO 关心商务过程，而测试人员必须在需求阶段规划测试用例。

原则 2：模型和代码应该更加紧密地联系在一起。如果操作系统是主要目标，那么任何对该操作系统不起作用的模型都是无用的。这就意味着必须一起声明代码和模型，可以使用工具来连接模型和代码。

原则 3：在模型和代码间建立双向信息流。当变更发生时，其必须可以在模型、代码和操作系统中传播。传统的代码变更反映在运行系统中，然而，在模型中反映代码变更也很重要。

原则 4：应该生成常规系统视图。系统元模型在 IT 管理层定义了商务过程和信息对象，在系统操作层定义了运行服务和物理节点，在软件工程层定义了需求视图。系统元模型的附属功能描述了从商业过程和商业对象到技术层的依赖性。

原则 5：模型信息必须持续跟踪系统变更。系统模型描述了所有抽象层的当前系统状态，把系统发展描述并记录为一系列系统模型快照。

原则 6：必须验证各个模型的信息一致性。模型持续验证和获得状态信息是支持利益相关者做出决策的两个重要服务需求。例如，软件架构师可能需要查看每个需求层的服务都有与之对应的架构层服务。

原则 7：每个模型元素都为利益相关者分配了权利和责任。每个利益相关者负责模型元素定义的一个子集，每个模型子集都是一位利益相关者关注的领域。这意味着每个模型元素可以为每位利益相关者描述这个元素执行的活动信息。

原则 8：应当描述各种模型元素的状态。因为在运行时关键变量控制的值定义了计算状态，所以分配给属性的值定义了每个模型元素的状态。

7.3.4 构建原则

构建活动包括一系列编码和测试任务，从而为向客户和最终用户交付可运行软件做好准备。在现代软件工程中，编码可能是：（1）直接生成编程语言源代码（例如 Java）；（2）使用待开发构件的类似设计的中间表示来自动生成源代码（例如企业架构^①）；或者（3）使用第四代编程语言（例如 Visual C#）自动生成可执行代码。

最初的测试是构件级的，通常称为单元测试。其他级别的测试包括：（1）集成测试（在构建系统的时候进行）；（2）确认测试——测试系统（或者软件增量部分）是否完全按照需求开发；（3）验收测试——由客户检验系统所有要求的功能和特性。在编码和测试过程中有一套原则，下面就讲述这些原则和概念。

编码原则。这些原则和概念与编程风格、编程语言和编程方法紧密结合。下面陈述一些基本的原则。

1. 准备原则。在写下每行代码之前，要确保：

引述 在生活中，我有软件窥探癖，常常偷看到别人很糟糕的代码。偶尔我也能发现一些真正有价值的东西——一些结构很好的代码，这些代码书写形式固定，与具体机器无关，每个构件都是那么简单、易于组织且易于修改。

David Parnas

① 企业构架是 Sparx 系统的工具，参考 <http://www.sparxsystems.com/products/ea/index.html>。

- 理解所要解决的问题。
- 理解基本的设计原则和概念。
- 选择一种能够满足构建软件以及运行环境要求的编程语言。
- 选择一种能提供工具以简化工作的编程环境。
- 构件级编码完成后进行单元测试。

2. 编程原则。在开始编码时，要确保：

- 遵循结构化编程 [Boh00] 方法来约束算法。
- 考虑使用结对编程。
- 选择能满足设计要求的数据结构。
- 理解软件架构并开发出与其相符的接口。
- 尽可能保持条件逻辑简单。
- 所开发的嵌套循环应易于测试。
- 选择有意义的变量名并符合相关编码标准。
- 编写注释，使代码具有自说明性。
- 增强代码的可读性（例如缩进和空行），使其有助于理解。

3. 确认原则。在完成第一阶段的编码之后，要确保：

- 适当进行代码走查。
- 进行单元测试并改正所发现的错误。
- 重构代码。

建议 避免开发一个解决错误问题的漂亮程序。特别要注意第一个准备原则。

122

讲述程序设计（编码）以及编码原则和概念的书比讲述软件过程其他论题的书要多很多。如 [Ker78] 的编程风格入门，[McC04] 的软件构造实践，[Ben99] 的编程珠玑，[Knu99] 的编程艺术，[Hun99] 的实用编程等很多主题的书。关于这些原则和概念的广泛讨论不在本书的范围内，如果你有兴趣，可参看一本或更多本上述参考书籍。

测试原则。在一本经典软件测试书中，Glen Myers[Mye79] 描述了一系列测试规则，这些规则很好地阐明了测试的目标：

- 测试是一个以查找程序错误为目的的程序执行过程。
- 好的测试用例能最大限度地找到尚未发现的错误。
- 成功的测试能找到那些尚未发现的错误。

这些目标意味着软件开发者在观念上的一些戏剧性的变化。他们持有与常人相反的观点——常人的观点是认为那些找不到一个错误的测试是成功的测试。我们的目标是要设计一些能用最短的时间、最少的工作量来系统地揭示不同类型错误的测试。

如果测试成功（按照前面论述的目标），就会发现软件中的错误。测试的第二个好处就是，它能表明软件功能的执行似乎是按照规格说明来进行的，行为需求和性能需求似乎也可以得到满足。此外，测试时收集的数据为软件的可靠性提供了很好的说明，并且从整体来看也为软件质量提供了一些说明。但是测试并不能说明某些错误和缺陷不存在，它只能显示出存在的错误和缺陷。在测试时保持这样一个观念是非常重要的，其实这并不是悲观。

网络资源 各种关于编码标准的链接可参看 <http://www.literateprogramming.com/links.html>。

提问 软件测试的目标是什么？

建议 在广泛的软件设计情景中，我们通常由着眼于软件架构的“宏观”层面开始，到着眼于构件模块的“微观”层面结束。对测试而言，着眼点和测试方式则正好相反。

Davis[Dav95b] 提出了一套测试原则^①，本书对这些原则做了一些改动。另外，Everett 和 Meyer[Eve09] 增加了一些原则。

原则 1：所有的测试都应该可以追溯到用户需求^②。软件测试的目标就是要揭示错误。而最严重的错误（从用户的角度来看）是那种导致程序无法满足需求的错误。

123

原则 2：测试计划应该远在测试之前就开始着手。测试计划（第 22 章）在分析模型一完成就应该开始。测试用例的详细定义可以在设计模型确定以后开始。因此，所有的测试在编码前都应该计划和设计好了。

原则 3：将 Pareto 原则应用于软件测试。简单地说，Pareto 原则认为在软件测试过程中 80% 的错误都可以在大概 20% 的程序构件中找到根源。接下来的问题当然就是要分离那些可疑的构件，然后对其进行彻底的测试。

原则 4：测试应该从“微观”开始，逐步转向“宏观”。最初计划并执行的测试通常着眼于单个程序模块，随着测试的进行，着眼点要慢慢转向在集成的构件簇中寻找错误，最后在整个系统中寻找错误。

原则 5：穷举测试是不可能的。即便是一个中等大小的程序，其路径排列组合的数目都非常庞大。因此，在测试中对每个路径组合进行测试是不可能的。然而，充分覆盖程序逻辑并确保构件级设计中的所有条件都通过测试是有可能的。

原则 6：为系统的每个模块做相应的缺陷密度测试。通常在最新的模块或者开发人员最缺乏理解的模块中进行这些测试。

原则 7：静态测试技术能得到很好的结果。有超过 85% 的软件缺陷源于软件文档（需求、规格说明、代码走查和用户手册）[Jon91]。这对系统文档测试是有价值的。

原则 8：跟踪缺陷，查找并测试未覆盖缺陷的模式。未发现的缺陷总数是软件质量好坏的指示器，未发现的缺陷类型可以很好地度量软件的稳定性。统计超时发现缺陷的模式可以预测缺陷的期望值。

原则 9：包含在演示软件中的测试用例是正确的行为。在维护和修改软件组件时，未预料到的交互操作会无意识地影响另外的一些组件。在软件产品变更后要准备检测系统行为，进行一组回归测试（第 22 章）是很重要的。

124

7.3.5 部署原则

正如我们在本书第一部分中提到的，部署活动包括三个动作：交付、支持和反馈。由于现代软件过程模型实质上是演化式或是增量式的，因此，部署活动并不是只发生一次，而是在软件完全开发完成之前进行许多次。每个交付周期都会向客户和最终用户提供一个可运行的并具有可用功能和特性的软件增量。每个支持周期都会为部署周期中所提到的所有功能和特性提供一些文档和人员帮助。每个反馈周期都会为软件开发团队提供一些重要的引导，以帮助修改软件功能、特性以及下一个增量所用到的方法。

软件增量交付对于任何一个软件项目来说都是一个重要的里程碑。以下将讲述一些软件开发团队在准备交付一个软件增量时所应该遵从的重要原则。

原则 1：客户对于软件的期望必须得到管理。客户期望的结果通常比软件团队承诺交付

① 这里给出的仅仅是 Davis 测试原则的一小部分。更多的信息参见 [Dav95b]。

② 这个原则是针对功能测试的，也就是说，测试重点在于需求。结构测试（测试重点在于体系结构的和逻辑的细节）可能没有直接涉及详细的需求。

的要多，这会很快令客户失望。这将导致客户反馈变得没有积极意义并且还会挫伤软件开发团队的士气。Naomi Karten[Kar94]在她的关于管理客户期望的书中提到：“管理客户期望首先应该认真考虑你该与客户交流什么与怎样交流。”她建议软件工程师必须认真地处理与客户有冲突的信息。（例如：对不可能在交付时完成的工作做出了承诺；在某次软件增量交付时交付了多于当初承诺要交付的工作，这将使得下次增量所要做的工作随之变少。^①）

建议 在提交软件增量前，要确保客户知道所期待的是什么。否则，客户期待的一定会超出你所提交的。

原则 2：完整的交付包应该经过安装和测试。所有可执行软件、支持数据文件、文档和一些相关的信息都必须组装起来，并经过实际用户的完整测试。所有的安装脚本和其他一些可操作的功能都应该在所有可能的计算配置（例如硬件、操作系统、外围设备、网络）环境中实施充分的检验。

原则 3：技术支持必须在软件交付之前就确定下来。最终用户希望在水问题发生时能得到及时的响应和准确的信息。如果技术支持跟不上或者根本就没有技术支持，那么客户会立即表示不满。支持应该是有计划的，准备好支持的材料并且建立适当的记录保持机制，这样软件开发团队就能按照支持请求种类进行分类评估。

125

原则 4：必须为最终用户提供适当的说明材料。软件开发团队交付的不仅仅是软件本身，也应该提供培训材料（如果需要的话）和故障解决方案，还应该发布关于“本次增量与以前版本有何不同”的描述。^②

原则 5：有缺陷的软件应该先改正再交付。迫于时间的压力，某些软件组织会交付一些低质量的增量，还会在增量中向客户提出警告：这些缺陷将在下次发布时解决。这样做是错误的。在软件商务活动中有这样一条谚语：“客户在几天后就会忘掉你所交付的高质量软件，但是他们永远忘不掉那些低质量的产品所出现的问题。软件会时刻提醒着问题的存在。”

已交付的软件会为最终用户提供一些好处，同时它也会为软件开发团队提供一些有用的反馈。当一个增量投入使用后，应该鼓励最终用户对软件的功能、特性以及易用性、可靠性和其他特性做出评价。

7.4 工作实践

Iskold [Isk08] 认为软件质量将成为软件公司之间竞争的分辨器。正如第 6 章所述，软件工程的人为因素和其他技术因素一样重要。因此，审视成功软件工程师的个性特征和工作习惯将会非常有趣。更重要的因素还包括：渴求不断重构设计代码，积极使用已验证的设计模式，在任何可能的情况下获取可复用的组件，关注可用性，开发可维护应用，采用对应用最好的编程语言，使用已验证的设计和测试实践来构建软件。

除了个性特征和工作习惯以外，Iskold [Isk08] 提出了 10 条观念，这些观念超越了编程语言和特殊技术。在软件过程中，有些观念是软件工程师提升自身价值的必要预备知识。

1. 接口。简洁熟悉的接口比复杂独特的接口少犯错误。
2. 习惯和模板。命名习惯和软件模板是大量软件开发人员和最终用户

引述 理想的软件工程师是复合型的……他不是科学家，不是数学家，不是社会学家或者作家；但是他或许会使用到任一或全部这些学科规范来解决软件问题。

N. W. Dougherty

126

^① 在交流活动中，软件团队应该确定用户希望的帮助材料是什么类型。

沟通的良好途径。

3. 层级法。层级法是数据和抽象编程的关键。它允许把设计理念和实施详情分离开，同时降低软件设计的复杂度。
4. 算法复杂性。从常规库中选择算法时，软件工程师必须领会算法的简洁和性能的特性。编写简洁可读的代码可以确保应用的时间和空间的有效性。
5. 哈希。哈希是有效的存储数据、抽取数据的重要方法。哈希也是在云数据库的计算机中进行数据定位的重要方法。
6. 缓存。软件工程师需要领会权衡关联关系，通过存储在计算机内存而不是第二个存储设备，提供快速进入数据集的方法，同时，当相互关联的数据没有在内存在中时，应用程序就会慢下来（例如实时视频游戏中的过场动画）。
7. 并发性。广为人知的多进程计算机和多线程编程环境的有效性将为软件工程带来挑战。
8. 云计算。云计算为各种类型的计算平台提供可靠且可访问的互联网服务和数据。
9. 安全性。每位计算机专业人士都应该关注保护系统设备的机密和健全性。
10. 关系数据库。关系数据库是信息存储和抽取的奠基石。知道如何减少数据冗余，以及如何提高抽取速度，这两点都是非常重要的。

在很多情况下，几个优秀软件工程师的“聪明才智”比数倍人的项目组更有效，优秀工程师必须知道使用哪些原则、实践和工具，并且在使用时知道为什么需要这些方法和工具。

7.5 小结

软件工程实践包括概念、原则、方法和在整个软件过程中所使用的工具。虽然每个软件工程项目是不同的，但却有着通用的普遍原则和一些与项目或产品无关的适用于每个过程框架活动的实践任务。

核心原则集有助于有意义的软件过程的应用以及有效的软件工程方法的执行。在过程级，核心原则建立了一个哲学基础，指导软件开发团队引导软件过程。在实践级，核心原则建立了一套价值准则，可以在分析问题、设计解决方案、实施方案、测试解决方案以及最终向用户部署软件时提供指导。 [127]

在开发者与客户进行沟通时，客户沟通原则主要着眼于两点：减少争吵和扩大双方的交流广度，双方必须互相协作以更好地交流。

策划原则着眼于为使开发整个系统或产品沿着最佳路线前进而提供指导。计划可以只是为某个软件增量而设计，或者为整个项目而制定。无论如何，计划都必须涉及要做什么、谁来完成以及什么时候完成。

建模包括分析和设计，描绘了逐渐细化的软件表示方法。建模的目的是加深对所要完成工作的理解，并为软件开发人员提供技术指导。建模原则是建立对软件进行表述的方法和注释的基础。

构建包括编码和测试循环，在这个循环中为每个构件生成源码并对其进行测试。编码原则定义了一些通用要求，在编码开始之前这些要求应该已经实现。尽管有许多的测试原则，但是只有一个是最主要的：测试是一个为了发现错误而执行程序的过程。

部署发生在向客户展示每个软件增量的时候，它包括交付、支持和反馈。交付的关键原则是管理客户期望并且能为客户提供合适的软件支持信息。支持需要预先准备。反馈允许客

户提出一些具有商业价值的变更意见，为开发者的下一个软件工程迭代周期提供输入。

习题与思考题

- 7.1 关注质量需要资源和时间，那么有可能既敏捷又维持对质量的关注吗？
- 7.2 在指导过程的 8 个核心原则中（在 7.2.1 节中讨论），你认为哪一个是最重要的？
- 7.3 用自己的语言描述关注点分解这个概念。
- 7.4 一个重要的沟通原则说到“有准备的沟通”。在早期的工作中你都需要做哪些准备？在早期的准备中都应该产生哪些工作产品？
- 7.5 研究沟通活动中的“协调人”，（使用提供的参考资料或其他资料）准备一套关于协调人的指南。
- 7.6 敏捷沟通与传统的软件工程沟通有什么不同？又有哪些相似之处？
- 7.7 “转换话题”为什么是必要的？
- 7.8 在沟通活动中需要做一些“协商”方面的调查研究，并且为“协商”准备一些指导方针。
- 7.9 请描述在项目进度表的上下文情景中粒度意味着什么。
- 7.10 为什么在软件工程中模型很重要？它们总是必需的吗？你对必要性的回答是否合格？
- 7.11 在设计建模的过程中有哪三个软件“域”需要考虑？
- 7.12 试着为 7.3.4 节中的编码添加一条附加的原则。
- 7.13 什么是成功的测试？
- 7.14 你是否同意下面的说法：“既然我们要向客户交付多个增量，那么为什么还要关注早期增量的质量——我们可以在今后的迭代中逐步改善这些问题。”说说你的看法。
- 7.15 为什么对于软件开发团队来说反馈是至关重要的？

扩展阅读与信息资源

在软件工程中，客户沟通是至关重要的活动，然而很少有团队愿意花时间阅读这方面的资料。Withall 的书（《Software Requirements, Patterns》，Microsoft Press, 2007）中提出了大量解决沟通问题的有用模式。Lamsweerd（《Requirement Engineering: From System Goals to UML Models to Software Specifications》，Wiley, 2009）和 Sutliff（《User-Centered Requirements》，Springer, 2002）非常重视与沟通相关的挑战。

Karten（《Changing How You Manage and Communicate Change》，IT Governance Publishing, 2009）、Weigers（《Software Requirements》，2nd ed., Microsoft Press, 2003）、Pardee（《To Satisfy and Delight Your Customer》，Dorset House, 1996）以及 Karten [Kar94] 都提供了许多关于客户交互的有效方法和观点。虽然并没有聚焦于软件，但 Hooks 和 Farry（《Customer Centered Products》，American Management Association, 2000）对如何与客户沟通提供了有用的通用指导方针。Young（《Project Requirements: A Guide to Best Practices》，Management Concepts, 2006；《Effective Requirements Practice》，Addison-Wesley, 2001）强调客户和开发人员组织“联合小组”协作进行需求分析。Hull、Jackson 和 Dick（《Requirements Engineering》，Springer, 3rd ed., 2010）以及 Somerville 和 Kotonya（《Requirements Engineering: Processes and Techniques》，Wiley, 1998）讨论“导入”概念、技术和其他需求工程原则。

沟通和策划的原则与概念在许多项目管理书籍中都有所涉及。一些比较有用的项目管理书籍包括：Juli（《Leadership Principles for Project Success》，CRC Press, 2012），West 和他的同事（《Project Management for IT Related Projects》，British Informatics Society, 2012），Wysocki（《Effective Project Management: Agile, Adaptive, Extreme》，5th ed., Wiley, 2009），Hughes（《Software

Project Management 》, 5th ed., McGraw-Hill, 2009), Bechtold (《 Essentials of Software Project Management 》, 2nd ed., Management Concept, 2007), Leach (《 Lean Project Management : Eight Principles for Success 》, BookSurge Publishing, 2006), 以及 Stellman 和 Green (《 Applied Software Project Management 》, O'Reilly Media, 2005)。

Davis[Dav95b] 编辑了一套出色的软件工程原则。此外, 每一本软件工程书中实际上都包括了对分析、设计和测试概念及原则的有益讨论。其中的佼佼者有(除了本书以外):

129

Abran, A. 和 J. Moore, 《 SWEBOK : Guide to the Software Engineering Body of Knowledge 》, IEEE, 2002。^①

Pfleeger, S., 《 Software Engineering: Theory and Practice 》, 4th ed., Prentice-Hall, 2009。

Schach, S., 《 Object-Oriented and Classical Software Engineering 》, McGraw-Hill, 8th ed., 2010。

Sommerville, I., 《 Software Engineering 》, 9th ed., Addison-Wesley, 2010。

这些书也对建模和创建原则做了详细的讨论。

建模原则可以参考很多着眼于需求工程和软件设计的书。Lieberman (《 The Art of Software Modeling 》, Auerbach, 2007)、Rosenberg 和 Stephen (《 Use Case Driven Object Modeling with UML : Theory and Practice 》, Apress, 2007)、Roques (《 UML in Practice 》, Wiley, 2004) 以及 Penker 和 Eriksson (《 Business Modeling with UML : Business Patterns at Work 》, Wiley, 2001) 都讨论了建模原则和方法。

Norman 的著作 (《 The Design of Everyday Things 》, Basic Books, 2002) 是每一个设计从业者的必读教材。Winograd 及其同事 (《 Bringing Design to Software 》, Addison-Wesley, 1996) 汇集了很多讨论软件设计实践的优秀文章。Constantine 和 Lockwood (《 Software for Use 》, Addison-Wesley, 1999) 提出了“以用户为中心的设计”相关的概念。Tognazzini (《 Tog on Software Design 》, Addison-Wesley, 1995) 提出了一些有价值的哲学讨论, 包括关于设计的本质、决策对于软件质量的影响以及团队能力对向客户提供软件的巨大价值影响。Stahl 及其同事 (《 Model-Driven Software Development : Technology, Engineering 》, Wiley, 2006) 讨论了模型驱动开发的原则。Halladay (《 Principle-Based Refactoring 》, Principle Publishing, 2012) 认为有 8 种基本设计原则和 50 条重构规则。

很多书籍都关注软件构建活动的一个或多个问题。Kernighan 和 Plauger[Ker78] 写了一本关于编码风格的经典书籍, McConnell[McC04] 提出了软件构建实践的实用指导原则, Bentley[Ben99] 提出了大量关于编码的宝贵建议, Knuth[Knu98] 写过一套关于编码艺术的三卷系列丛书, Hunt[Hun99] 提出了实用的编程原则。

Myers 及其同事 (《 The Art of Software Testing 》, 3rd ed., Wiley, 2011) 主要修订了他的经典著作版本, 并且讨论了许多重要的测试原则。《 How Google Tests Software 》一书 (Addison-Wesley, 2012)、Perry (《 Effective Methods for Software Testing 》, 3rd ed., Wiley, 2006)、Whittaker (《 How to Break Software 》, Addison-Wesley, 2002)、Kaner 和他的同事 (《 Lessons Learned in Software Testing 》, Wiley, 2001) 以及 Marick (《 The Craft of Software Testing 》, Prentice-Hall, 1997) 都提出了重要的测试概念、原则和很多实用指导。

网上有大量关于软件工程实践的信息资源, 与软件工程实践相关的实时更新参考文献见 SEPA 网站 www.mhhe.com/pressman。

130

① 免费获得: <http://www.computer.org/portal/web/swebok/v3guide>。

理解需求

要点浏览

概念：在开始任何技术工作之前，关注于一系列需求工程任务都是个好主意。这些任务有助于你理解软件将如何影响业务、客户想要什么以及最终用户将如何和软件交互。

人员：软件工程师（在IT界有时指系统工程师或分析员）以及项目的其他利益相关者（项目经理、客户、最终用户）都将参与需求工程。

重要性：在设计和开发某个优秀的计算机软件时，如果软件解决的问题是错误的，那么即使软件再精巧也满足不了任何人的要求。这就是在设计和开发一个基于计算机的系统之前理解客户需求的重要性所在。

步骤：需求工程首先是起始阶段（定义将要解决的问题的范围和性质）；然后是获

取（帮助利益相关者定义需要什么）；接下来是细化（精确定义和修改基本需求）。当利益相关者提出问题后，就要进行协商（如何定义优先次序？什么是必需的？何时需要？）最后，以某种形式明确说明问题，再经过评审或确认以保证我们和利益相关者对于问题的理解是一致的。

工作产品：需求工程的目的在于为各方提供关于问题的一份书面理解。不过依然可以得到工作产品：用户场景，功能和特性列表，需求模型或规格说明。

质量保证措施：利益相关者评审需求工程的工作产品，以确保你所理解的正是他们真正想要的。需要提醒大家的是：即使参与各方均认可，事情也会有变化，而且变化可能贯穿整个项目实施过程。

理解问题的需求是软件工程师所面对的最困难的任务之一。第一次考虑这个问题时，开发一个清晰且易于理解的需求看起来似乎并不困难。毕竟，客户难道不知道需要什么？最终用户难道对将给他们带来实际收益的特性和功能没有清楚的认识？不可思议的是，很多情况下的确是这样的。甚至即使用户和最终用户清楚地知道他们的要求，这些要求也会在项目的实施过程中改变。

在 Ralph Young[You01] 的一本关于有效需求实践的书的前言中，我写道：

这是最恐怖的噩梦：一个客户走进你的办公室，坐下，正视着你，然后说：“我知道你认为你理解我说的是是什么，但你并不理解的是，我所说的并不是我想要的。”这种情况总是在项目后期出现，而当时的情况通常是：已经做出交付期限的承诺，声誉悬于一线并且已经投入大量资金。

关键概念

- 分析模式
- 协作
- 细化
- 获取
- 起始
- 协商
- 质量功能部署
- 需求工程
- 需求收集
- 需求管理
- 需求监控
- 规格说明
- 利益相关者

我们这些已经在系统和软件业中工作多年的人就生活这样的噩梦中，而且目前还都不知道该怎么摆脱。我们努力从客户那里获取需求，但却难以理解获取的信息。我们通常采用混乱的方式记录需求，而且总是花费极少的时间检验到底记录了什么。我们容忍变更控制自己，而不是建立机制去控制变更。总之，我们未能为系统或软件奠定坚实的基础。这些问题中的每一个都是极富挑战性的，当这些问题集中在一起时，即使是最有经验的管理人员和实际工作人员也会感到头痛。但是，确实存在解决方法。

把需求工程称作以上所述挑战的“解决方案”可能并不合理，但需求工程确实为我们提供了解决这些挑战的可靠途径。

8.1 需求工程

设计和编写软件富有挑战性、创造性和趣味性。事实上，编写软件是如此吸引人，以至于很多软件开发人员在清楚地了解用户需要什么之前就迫切地投入到编写工作中。开发人员认为：在编写的过程中事情总是会变得清晰；只有在检验了软件的早期版本后项目利益相关者才能够更好地理解要求；事情变化太快，以至于理解需求细节是在浪费时间；最终要做的是开发一个可运行的程序，其他都是次要的。构成这些论点的原因在于其中也包含了部分真实情况^①，但是这中间的每个论点都存在一些小问题，汇集在一起就可能导致软件项目的失败。

需求工程（Requirement Engineering, RE）是指致力于不断理解需求的大量任务和技术。从软件过程的角度来看，需求工程是一个软件工程动作，开始于沟通并持续到建模活动。它必须适用于过程、项目、产品和人员的需要。

需求工程在设计和构建之间建立起联系的桥梁。桥梁源自何处？有人可能认为源于项目利益相关者（如项目经理、客户、最终用户），也就是在他们那里定义业务需求、刻画用户场景、描述功能和特性、识别项目约束条件。其他人可能会建议从宽泛的系统定义开始，此时软件只是更大的系统范围中的一个构件。但是不管起始点在哪里，横跨这座桥梁都将把我们带到项目之上更高的层次：允许由软件团队检查将要进行的软件工作的内容；必须提交设计和构建的特定要求；完成指导工作顺序的优先级定义；以及将深切影响随后设计的信息、功能和行为。

在过去的几十年，有很多技术变革影响着需求工程的过程 [Wev11]。无处不在的计算使计算机技术能够与许多日常事务相结合。这些事务通过联网就能生成更多完整的用户信息，同时伴随着对隐私和安全问题的关注。

在电子市场广泛传播的应用引领了更多各式各样利益相关者的需求。利益相关者能定制产品，以满足一小部分最终用户特定的需求目标。当产品开发周期缩短时，流水线型需求工程会有压力，目的是推动产品更快进入市场。但是存在

关键概念

利益相关者
用例
确认
观点
工作产品

引述

构建一个软件系统最困难的部分是确定构建什么。其他工作不会像这部分工作一样，在出错之后会如此严重地影响随后实现的系统，并且在以后修补竟会如此困难。

Fred Brooks

关键点

需求工程为设计和构造奠定了坚实的基础。如果没有需求工程，那么实现的软件很有可能无法满足客户的需求。

建议

可以在需求阶段做一些设计工作，在设计阶段做一些需求工作。

132

① 对小项目（不超过一个月）和只涉及简单的软件工作的更小项目，这确实是正确的。但随着软件规模和复杂性的增加，这些论点就开始出问题了。

的根本问题仍然是如何及时获得精准稳定的利益相关者的输入信息。

需求工程包括七项明确的任务：起始，获取，细化，协商，规格说明，确认和管理。注意，这些需求工作中的一些任务会并行发生，并且要全部适应项目的要求。

起始。如何开始一个软件项目？有没有一个独立的事件能够成为新的基于计算机的系统或产品的催化剂？需求会随时间的流逝而发展吗？这些问题没有确定的答案。某些情况下，一次偶然的交谈就可能导致大量的软件工程工作。但是多数项目都是在确定了商业要求或是发现了潜在的新市场、新服务时才开始。业务领域的利益相关者（如业务管理人员、市场人员、产品管理人员）定义业务用例，确定市场的宽度和深度，进行粗略的可行性分析，并确定项目范围的工作说明。所有这些信息都取决于变更，但是应该充分地^①与软件工程组织^②及时讨论。在项目起始阶段中^③，要建立基本的理解，包括存在的问题、谁需要解决方案、所期望解决方案的性质、与项目利益相关者和开发人员之间达成初步交流合作的效果。

引述 大多数软件灾难的种子通常都是在软件项目开始的头三个月内种下的。

Caper Jones

获取。询问客户、用户和其他人：系统或产品的目标是什么，想要实现什么，系统和产品如何满足业务的要求，最终系统或产品如何用于日常工作。这些问题看上去是非常简单的，但实际上并非如此，而是非常困难。

获取过程中最重要的是建立商业目标 [Cle10]。我们的工作就是与利益相关者约定，鼓励他们诚实地分享目标。一旦抓住目标，就应该建立优先机制，并（为满足利益相关者的目标）建立潜在架构的合理性设计。

信息栏 基于目标导向的需求工程

目标是一个系统或产品必须达到的长期目的。目标可能涉及功能性或非功能性（例如可靠性、安全性、可用性等）内容。目标通常是向利益相关者解释需求的好方法，一旦建立了目标，就可以在利益相关者中处理冲突和矛盾。

从目标中可以系统地推出目标模型（第10章和第11章）和需求。展示目标之间各种链接的目标图，能提供一定程度的追踪性（8.2.6节），从高层次的策略关

注到低层次的技术细节。目标应该精确定义，并为需求的细化、验证与确认、冲突管理、协商、解释和发展提供服务基础。

需求检测中的冲突通常是目标自身存在冲突的结果。通过一套相互商定的目标可获得冲突解决方案，这些目标与每个成员及利益相关者的渴求相一致。有关目标和需求工程更完备的讨论可以查找 Lamsweweerde 的论文 [LaM01b]。

Christel 和 Kang[Cri92] 指出了一系列问题，可以帮助我们理解为什么获取需求这么困难。范围问题发生在系统边界不清楚的情况下，或是客户和用户的说明带有不必要的技术细节，这些细节可能会导致混淆而不是澄清系统的整体目标。理解问题发生在客户和用户并不

① 如果要开发一个基于计算机的系统，那么讨论将从系统工程过程开始。请访问 www.mhhe.com/pressman 获取更多系统工程的讨论详情。

② 应该记得（第4章）统一过程定义了更全面的“起始阶段”，包括本章所讨论的起始、获取和细化工作。

完全确定需要什么的情况下,包括:对其计算环境的能力和限制所知甚少,对问题域没有完整的认识,与系统工程师在沟通上存在问题,忽略了那些他们认为是“明显的”信息,确定的需求和其他客户及用户的要求相冲突,需求说明有歧义或不可测试。易变问题发生在需求随时间推移而变更的情况下。为了帮助解决这些问题,需求工程师必须以有组织的方式开展需求收集活动。

细化。在起始和获取阶段获得的信息将在细化阶段进行扩展和提炼。该任务的核心是开发一个精确的需求模型(第9~11章),用以说明软件的功能、特征和信息的各个方面。

细化是由一系列的用户场景建模和求精任务驱动的。这些用户场景描述了如何让最终用户和其他参与者与系统进行交互。解析每个用户场景以便提取分析类——最终用户可见的业务域实体。应该定义每个分析类的属性,确定每个类所需要的服务^①,确定类之间的关联和协作关系,并完成各种补充图。

协商。业务资源有限,而客户和用户却提出了过高的要求,这是常有的事。另一个相当常见的现象是,不同的客户或用户提出了相互冲突的需求,并坚持“我们的特殊要求是至关重要的”。

需求工程师必须通过协商过程来调解这些冲突。应该让客户、用户和其他利益相关者对各自的需求排序,然后按优先级讨论冲突。使用迭代的方法给需求排序,评估每项需求的成本和风险,处理内部冲突,删除、组合或修改需求,以便参与各方均能达到一定的满意度。

规格说明。在基于计算机的系统(和软件)的环境下,术语规格说明对不同的人有不同的含义。规格说明可以是一份写好的文档、一套图形化的模型、一个形式化的数学模型、一组使用场景、一个原型或上述各项的任意组合。

有人建议应该开发一个“标准模板”[Som97]并将之用于规格说明,他们认为这样将促使以一致的从而也更易于理解的方式来表示需求。然而,在开发规格说明时保持灵活性有时是必要的,对大型系统而言,文档最好采用自然语言描述和图形化模型来编写。而对于技术环节明确的较小产品或系统,使用场景可能就足够了。

提问 为什么获得对客户需要的清晰理解会非常困难?

134

建议 细化是件好事,但你必须知道何时可以停止细化。关键是能采用为设计建立一个坚实基础的方式说明问题。如果超出这个点就是在做设计。

建议 在有效的协商中没有赢家也没有输家,而是双赢。这是因为双方合作才是“交易”的坚实基础。

关键点 规格说明的形式和规格随着待开发软件的规模和复杂度的不同而变化。

135

信息栏 软件需求规格说明模板

软件需求规格说明(SRS)是在项目商业化之前必须建立的详细描述软件各个方面的工作产品。值得注意的是,常常没有正规的SRS。事实上很多实例表明,在软件需求规格说明上投入的工作量还不如投入到其他软件工程活动中。然而,如果

软件由第三方开发,当缺少规格说明导致严重业务问题时,或是当系统非常复杂或业务十分重要时,才能表明需求规格说明是非常必要的。

Process Impact公司的Karl Wiegers [Wie03]开发了一套完整的模板(参考

① 服务通过对类的封装操作数据,也可使用术语“操作”和“方法”。如果你不熟悉面向对象的概念,附录2有基本的人门指导。

www.processimpact.com/process_assets/srs_template.doc), 能为那些必须建立完整需求规格说明书的人提供指导。主题大纲如下:

目录

版本历史

1. 引言

1.1 目的

1.2 文档约定

1.3 适用人群和阅读建议

1.4 项目范围

1.5 参考文献

2 总体描述

2.1 产品愿景

2.2 产品特性

2.3 用户类型和特征

2.4 操作环境

2.5 设计和实施约束

2.6 用户文档

2.7 假设和依赖

3. 系统特性

3.1 系统特性 1

3.2 系统特性 2(等等)

4 外部接口需求

4.1 用户接口

4.2 硬件接口

4.3 软件接口

4.4 通信接口

5. 其他非功能需求

5.1 性能需求

5.2 安全需求

5.3 保密需求

5.4 软件质量属性

6. 其他需求

附录 A: 术语表

附录 B: 分析模型

附录 C: 问题列表

每个需求规格说明主题的详细描述可以从前面所提的 URL 下载 SRS 模板来得到。

确认。在确认这一步将对需求工程的工作产品进行质量评估。需求确认要检查规格说明^①以保证: 已无歧义地说明了所有的系统需求; 已检测出不一致性、疏忽和错误并予以纠正; 工作产品符合为过程、项目和产品建立的标准。

正式的技术评审(第 20 章)是最主要的需求确认机制。确认需求的评审小组包括软件工程师、客户、用户和其他利益相关者, 他们检查系统规格说明, 查找内容或解释上的错误, 以及可能需要进一步澄清的地方、丢失的信息、不一致性(这是建造大型产品或系统时遇到的主要问题)、冲突的需求或是不可实现的(不能达到的)需求。

为了说明发生在需求验证过程中的某些问题, 要考虑两个看似无关紧要的需求:

- 软件应该对用户友好。
- 成功处理未授权数据库干扰的比率应该小于 0.0001。

第一个需求对开发者而言概念太模糊, 以至于不能测试或评估。什么是“用户友好”的精确含义? 为了确认它, 必须以某种方式使其量化。

第二个需求是一个量化元素(“小于 0.0001”), 但干扰测试会很困难且很费时。这种级别的安全真的能保证应用系统吗? 其他附加的与安全相关的需求(例如密码保护、特定的握

建议 需求确认时的一个重要问题是一致性。使用分析模型可以保证需求说明的一致性。

① 每个项目有不同的规格说明特性。在某些情况下, 规格说明是指收集到的用户场景或其他一些事物。在另一些情况下, 规格说明可以包括用户场景、模型和说明性文档。

手协议)能代替指明的定量需求吗?

Glinz[Gli09] 写到质量需求要以恰当的方式表述,从而保证交付最优价值。这意味着要对不能满足利益相关者质量要求的交付系统进行风险(第35章)评估,并且试图以最小代价减轻风险。质量需求越关键,越需要采用量化术语来陈述。在某些情况下,常见质量需求可以使用定性技术进行验证(例如用户调查或检查表)。在其他情况下,质量需求可以使用定性和定量相结合的评估方式进行验证。

信息栏 需求确认检查单

通常,按照检查单上的一系列问题检查每项需求是非常有用的。这里列出其中部分可能会问到的问题:

- 需求说明清晰吗?有没有可能造成误解?
- 需求的来源(如人员、规则、文档)弄清了吗?需求的最终说明是否已经根据或对照最初来源检查过?
- 需求是否用定量的术语界定?
- 其他哪些需求和此需求相关?是否已经使用交叉索引矩阵或其他机制清楚地加以说明?
- 需求是否违背某个系统领域的约束?
- 需求是否可测试?如果可以,能否说明检验需求的测试(有时称为确认准则)?
- 对已经创建的任何系统模型,需求是否可追溯?
- 对整体系统/产品目标,需求是否可追溯?
- 规格说明的构造方式是否有助于理解、轻松引用和翻译成更技术性的工作产品?
- 对已创建的规格说明是否建立了索引?
- 和系统性能、行为及运行特征相关的需求说明是否清楚?哪些需求是隐含出现的?

137

需求管理。对于基于计算机的系统,其需求会变更,而且变更的要求贯穿于系统的整个生命周期。需求管理是用于帮助项目组在项目进展中标识、控制和跟踪需求以及需求变更的一组活动^①。这类活动中的大部分和第29章中讨论的软件配置管理(SCM)技术是相同的。

软件工具 需求工程

[目标]需求工程工具有助于需求收集、需求建模、需求管理和需求确认。

[机制]工具的工作机制多种多样。通常,需求工程工具创建大量的图形化(例如UML)模型用以描述系统的信息、功能和行为。这些模型构成了软件过程中其他所有活动的基础。

[代表性工具]^②

Volere 需求资源网站 www.volere.co.uk/tools.htm 提供了一个相当全面(也是最新)的需求工程工具列表。我们将在第9、10章讨论需求建模工具。下面提到的工具主要侧重于需求管理。

- EasyRM。由 Cybernetic Intelligence GmbH

① 正规的需求管理只适用于具有数百个可确认需求的大型项目。对于小项目,该需求工程工作可以适当裁剪,一定程度的不正规也是可以接受的。

② 这里提到的工具只是此类工具的例子,并不代表本书支持采用这些工具。在大多数情况下,工具名被各自的开发者注册为商标。

开发 (<http://www.visuresolutions.com/visure-requirement-software>), 可视化需求是一套灵活完整的需求工程生命周期解决方案, 支持需求捕获、分析、规格说明、确认和验证、管理和复用。

- Rational RequisitePro。由 Rational 软件开发 (www-03.ibm.com/software/products/

us/en/reqpro/), 允许用户建立需求数据库, 表述需求之间的关系, 并且组织、排序和跟踪需求。

从前面所提的 Volere 网站和 www.jiludwig.com/requirements_management_tools.html 还可以找到很多额外的需求管理工具。

8.2 建立根基

在理想情况下, 利益相关者和软件工程师在同一个小组中工作^①。在这种情况下, 需求工程就只是和组里熟悉的同事进行有意义的交谈。但实际情况往往不是这样。

138

客户或最终用户可能位于不同的城市或国家, 对于想要什么可能仅有模糊的想法, 对于将要构建的系统可能存在有冲突的意见, 他们的技术知识可能很有限, 而且只有有限的时间与需求工程师沟通。这些事情都是不希望遇到的, 但却又是十分常见的, 软件开发团队经常被迫在这种环境的限制下工作。

下节将要讨论启动需求工程所必需的步骤, 以便理解软件需求, 使得项目自始至终走向成功解决方案的方向。

8.2.1 确认利益相关者

Sommerville 和 Sawyer[Som97] 把利益相关者定义为“直接或间接地从正在开发的系统中获益的人”。可以确定如下几个容易理解的利益相关者: 业务运行管理人员、产品管理人员、市场销售人员、内部和外部客户、最终用户、顾问、产品工程师、软件工程师、支持和维护工程师以及其他人员。每个利益相关者对系统都有不同的考虑, 当系统成功开发后所能获得的利益也不相同, 同样, 当系统开发失败时所面临的风险也是不同的。

建议 利益相关者是那些对将要开发的系统有直接的兴趣或直接从中获益的人。

在开始阶段, 需求工程师应该创建一个人员列表, 列出那些有助于获取需求的人员(8.3节)。最初的人员列表将随着接触的利益相关者人数的增多而增加, 因为每个利益相关者都将被询问“你认为我还应该和谁交谈”。

8.2.2 识别多重观点

因为存在很多不同的利益相关者, 所以系统需求调研也将从很多不同的视角开展。例如, 市场销售部门关心能激发潜在市场的、有助于新系统销售的功能和特性; 业务经理关注应该在预算内实现的产品特性, 并且这些产品特性应该满足已规定的市场限制; 最终用户可能希望系统的功能是他们所熟悉的并且易于学习和使用; 软件工程师可能关注非技术背景的利益相关者看不到的软件基础设施, 使其能够支持更多的适于销售的功能和特性; 支持工程师可能关注软件的可维护性。

引述 把三个利益相关者请进一个房间, 然后问他们想要什么样的系统, 你很可能会得到四个或更多的不同观点。

作者不详

① 推荐所有项目都使用该方法, 而且该方法是敏捷软件开发方法的主要部分。

这些参与者（以及其他人员）中的每一个人都将为需求工程贡献信息。当从多个角度收集信息时，所形成的需求可能存在不一致性或相互矛盾。需求工程师的工作就是把所有利益相关者提供的信息（包括不一致或是矛盾的需求）分类，分类的方法应该便于决策制定者为系统选择一个内部一致的需求集合。

为使软件满足用户而获取需求的过程存在很多困难：项目目标不清晰，利益相关者的优先级不同，人们还没说出的假设，相关利益者解释含义的不同，很难用一种方式对陈述的需求进行验证 [Ale11]。有效需求工程的目标是去除或尽力减少这些问题的发生。

139

8.2.3 协同合作

假设一个项目中有五个利益相关者，那么对一套需求就会有五种或更多的正确观点。在前面几章中我们已经注意到客户（和其他利益相关者）之间应该团结协作（避免内讧），并和软件工程人员团结协作，这样才能成功实现预定的系统。但是如何实现协作？

需求工程师的工作是标识公共区域（即所有利益相关者都同意的需求）和矛盾区域（或不一致区域，即某个利益相关者提出的需求和其他利益相关者的需求相矛盾）。当然，后一种矛盾区域的解决更有挑战性。

信息栏 使用“优先点”

有一个方法能够解决相互冲突的需求，同时更好地理解所有需求的相对重要性，那就是使用基于“优先点”的“投票”方案。所有的利益相关者都会分配到一定数量的优先点，这些优先点可以适用于很多需求。在需求列表上，每个利益相关者通过向每个需求分配一个或多个优先点来

表明（从他的个人观点）该需求的相对重要性。优先点用过之后就不能再次使用，一旦某个利益相关者的优先点用完，他就不能再对需求实施进一步的操作。所有利益相关者在每项需求上的优先点总数显示了该需求的综合重要性。

协作并不意味着必须由委员会定义需求。在很多情况下，利益相关者的协作是提供他们各自关于需求的观点，而一个有力的“项目领导者”（例如业务经理或高级技术员）可能要对删减哪些需求做出最终判断。

8.2.4 首次提问

在项目开始时的提问应该是“与环境无关的” [Gau89]。第一组与环境无关的问题集中于客户和其他利益相关者以及整体目标和收益。例如，需求工程师可能会问：

- 谁是这项工作的最初请求者？
- 谁将使用该解决方案？
- 成功的解决方案将带来什么样的经济收益？
- 对于这个解决方案你还需要其他资源吗？

这些问题有助于识别所有对构建软件感兴趣的利益相关者。此外，问题还确认了某个成功实现的可度量收益以及定制软件开发的可选方案。

引述 知道问题比知道所有的答案更好。

James Thurber

140

下一组问题有助于软件开发组更好地理解问题，并允许客户表达其对解决方案的看法：

- 如何描述由某成功的解决方案产生的“良好”输出的特征？
- 该解决方案强调解决了什么问题？
- 能向我们展示（或描述）解决方案使用的商业环境吗？
- 存在将影响解决方案的特殊的性能问题或约束吗？

最后一组问题关注沟通活动本身的效率。Gause 和 Weinberg[Gau89]称之为“元问题”。下面给出了元问题的简单列表：

- 你是回答这些问题的合适人选吗？你的回答是“正式的”吗？
- 我的提问和你想解决的问题相关吗？
- 我的问题是否太多了？
- 还有其他人员可以提供更多的信息吗？
- 还有我应该问的其他问题吗？

这些问题（和其他问题）将有助于“打破坚冰”，并有助于交流的开始，而且这样的交流对成功获取需求至关重要。但是，会议形式的问与答（Q&A）并非是一定会取得成功的好方法。事实上，Q&A 会议应该仅仅用于首次接触，然后应该用问题求解、协商和规格说明等需求获取方式来取代。在 8.3 节中将介绍这类方法。

提问 什么问题有助于你获得对问题的初步认识？

引述 问问题的人是五分钟的傻瓜，而不问问题的人将永远是傻瓜。

中国谚语

8.2.5 非功能需求

可以将非功能需求（Nonfunctional Requirement, NFR）描述成质量属性、性能属性、安全属性或一个系统中的常规限制。利益相关者通常不易清晰地表述这些内容。Chung[Chu09]提醒我们确实有片面强调软件功能的情况，然而如果没有必要的非功能属性，那么软件将无法使用。

141

在 8.3.2 节中，我们将讨论质量功能部署（Quality Function Deployment, QFD）。质量功能部署试图将客户未说出的要求或目标转换成系统需求。在软件需求规格说明书中常常单独列出非功能需求。

作为 QFD 的附属，尽可能定义一个两阶段方法 [Hne11]，以帮助软件团队和利益相关者识别非功能需求。在第一阶段为系统建立一套软件工程指南，其中包括最佳实践指南，还表述了架构风格（第 13 章）和设计模式（第 16 章）的应用。然后开发一套 NFR（例如可用性、可测性、安全性或可维护性）的列表。在这个简单的表格中，列项表示 NFR，行项表示软件工程指南。关系矩阵把每条指南与其他指南相对比，帮助团队评估每对指南是否完备、重叠、冲突或独立。

在第二阶段，团队根据一套决策规则 [Hne11]（用来决定实施哪些指南、放弃哪些指南）划分出同类的非功能需求，并为其排出优先级。

8.2.6 可追溯性

可追溯性是软件工程用语，指在软件工作产品间记录的链接（例如需求和测试用例），需求工程师用可追溯矩阵表达需求和其他软件工程产品间的相互关系。用需求名称标识可追溯矩阵的行，用软件工作产品的名称标识可追溯矩阵的列（例如设计元素或测试用例）。矩阵的单元格表示两者链接的百分比。

可追溯矩阵支持各种工程开发活动。无论采用哪个过程模型，它都能为开发者提供项目从一个阶段到另一个阶段的连续性。可追溯矩阵通常可用于确保工程中的工作产品考虑了所有需求。

随着需求和工作产品数量的增长，保持可追溯矩阵的更新会随之变得困难。虽然如此，但跟踪产品需求的影响和发展还是会产生一定的重要意义 [Got11]。

8.3 获取需求

需求获取（又称为需求收集）将问题求解、细化、协商和规格说明等方面的元素结合在一起。为了鼓励合作，一个包括利益相关者和开发人员的团队共同完成如下任务：确认问题，为解决方案的相关元素提供建议，商讨不同的方法并描述初步的需求解决方案 [Zah90]。^①

[142]

8.3.1 协作收集需求

关于需求收集，现在已经提出了很多不同的协同需求收集方法，各种方法适用于稍有不同的场景，而且所有这些均是在下面的基本原则之上做了某些改动：

- 实际的或虚拟的会议由软件工程师和其他利益相关者共同举办和参与。
- 制定筹备和参与会议的规则。
- 建议拟定一个会议议程，这个议程既要足够正式，使其涵盖所有的要点，但也不能太正式，以鼓励思维的自由交流。
- 由一个“主持人”（可以是客户、开发人员或其他人）控制会议。
- 采用“方案论证手段”（可以是工作表、活动挂图、不干胶贴纸、电子公告牌、聊天室或虚拟论坛）。

提问 主持一个协作的需求收集会议的基本原则是什么？

网络资源 联合应用程序开发 (JAD) 是需求收集普遍采用的技术。可以从以下地址找到详细的说明：www.carolla.com/wp-jad.htm。

协作收集需求的目标是标识问题，提出假设解决方案的相关元素，协商不同方法以及确定一套解决需求问题的初步方案。

在需求的起始阶段写下 1~2 页的“产品要求”（8.2 节），选择会议地点、时间和日期，选派主持人，邀请软件团队和其他利益相关者参加会议。在会议日期之前，给所有参会者分发产品需求。

举一个例子^②，考虑 SafeHome 项目中的一个市场营销人员撰写的产品要求。此人对 SafeHome 项目的住宅安全功能叙述如下：

我们的研究表明，住宅管理系统市场以每年 40% 的速度增长。我们推向市场的首个 SafeHome 功能将是住宅安全功能，因为多数人都熟悉“报警系统”，所以这 will 更容易销售。

住宅安全功能应该为防止和识别各种不希望出现的“情况”提供保护，如非法入侵、火灾、漏水、一氧化碳浓度超标等。该功能将使用无线传感器监控每种情况，户主可以用程序控制，并且在出现状况时系统将自动用电话联系监控部门。

[143]

事实上，其他人在需求收集会议中将补充大量的信息。但是，即使有了补充信息，仍有可能存在歧义性和疏漏，也有可能发生错误。但在目前的情况下，上面的“功能描述”是足

① 这种方法有时称为协助应用规格说明技术 (Facilitated Application Specification Technique, FAST)。

② SafeHome（有一定的扩展和变动）在下面很多章节中用于说明软件工程的重要方法。作为一个练习，为该例子举行你自己的需求收集会议并为其开发一组列表是值得的。

够的。

在召开会议评审产品要求的前几天，要求每个与会者列出构成系统周围环境的对象、由系统产生的其他对象以及系统用来完成功能的对象。此外，要求每个与会者列出服务操作或与对象交互的服务（过程或功能）列表。最后，还要开发约束列表（如成本、规模大小、业务规则）和性能标准（如速度、精确度）。告诉与会者，这些列表不要求完备无缺，但要反映每个人对系统的理解。

SafeHome 描述的对象可能包括：一个控制面板、若干烟感器、若干门窗传感器、若干动态检测器、一个警报器、一个事件（一个已被激活的传感器）、一台显示器、一台计算机、若干电话号码、一个电话等。服务列表可能包括：配置系统、设置警报器、监控传感器、电话拨号、控制面板编程以及读显示器（注意，服务作用于对象）。采用类似的方法，每个与会者都将开发约束列表（例如，当传感器不工作时系统必须能够识别，必须是用户友好的，必须能够和标准电话线直接连接）和性能标准列表（例如，一个传感器事件应在一秒内被识别，应实施事件优先级方案）。

这些对象列表可以用一张大纸钉在房间的墙上或用便签纸贴在墙上或写在墙板上。或者，列表也可以被贴在内部网站的电子公告牌上或聊天室中，便于会议前的评审。理想情况下，应该能够分别操作每个列表项，以便于合并列表、删除项以及加入新项。在本阶段，严禁批评和争论。

当某一专题的各个列表被提出后，小组将生成一个组合列表。该组合列表将删除冗余项，并加入在讨论过程中出现的一些新的想法，但是不删除任何东西。在所有专题的组合列表都生成后，由主持人组织开始讨论。组合列表可能会缩短、加长或重新措词，以求更恰当地反映即将开发的产品或系统，其目标是为所开发系统的对象、服务、约束和性能提交一个意见一致的列表。

在很多情况下，列表所描述的对象或服务需要更多的解释。为了完成这一任务，利益相关者为列表中的条目编写小规格说明（mini-specification），或者生成包括对象或服务的用户用例（8.4节）。例如，对 SafeHome 对象控制面板的小规格说明如下：

控制面板是一个安装在墙上的装置，尺寸大概是 230mm×130mm。控制面板与传感器、计算机之间无线连接，通过一个 12 键的键盘与用户交互，通过一个 75mm×75mm 的 OLED 彩色显示器为用户提供反馈信息。软件将提供交互提示、回显以及类似的功能。

然后，将每个小规格说明提交给所有利益相关者进行讨论，进行添加、删除和进一步细化等工作。在某些情况下，编写小规格说明可能会发现新的对象、服务、约束或性能需求，可以将这些新发现加入到原始列表中。在所有的讨论过程中，团队可能会提出某些在会议中不能解决的问题，将这些问题列表保留起来以便这些意见在以后的工作中发挥作用。

建议 如果一个系统或产品将要为很多用户提供服务，那么必须绝对保证需求是从所有用户的代表群中提取的。如果只有一个用户定义所有的需求，那么接受风险将会非常高。

引述 事实不会因为被忽略而不存在。

Aldous Huxley

建议 一定要避免攻击客户意见“太昂贵”或“不实际”这样的严厉谴责。这里建议通过协商形成一个大家均接受的列表。为了达到这个目标，必须保持开放的思想。

144

SafeHome 召开需求收集会议

[场景] 一间会议室，进行首次需求收集会议。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Doug Miller，

软件工程师；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人（指向白板）：这是目前住宅安全功能的对象和服务列表。

营销人员：从我们的观点看差不多覆盖了需求。

Vinod：没有人提到他们希望通过 Internet 访问所有的 SafeHome 功能吗？这应该包括住宅安全功能，不是吗？

营销人员：是的，这很正确……我们必须加上这个功能以及合适的对象。

主持人：这还需要加上一些限制吗？

Jamie：肯定，包括技术上的和法律上的。

产品代表：什么意思？

Jamie：我们必须确保外人不能非法侵入系统、使系统失效、抢劫甚至更糟。我们的责任非常重。

Doug：非常正确。

营销人员：但我们确实需要……只是保证能够制止外人进入……

Ed：说比做起来容易，而且……

主持人（打断）：我现在不想讨论这个问题。我们把它作为动作项记录下来，然后继续讨论（Doug 作为会议的记录者记下合适的内容）。

支持人：我有种感觉，这儿仍存在很多需要考虑的问题。

（小组接下来花费 20 分钟提炼并扩展住宅安全功能的细节。）

许多利益相关者关心（例如准确率、数据可用性、安全性）这些基本的非功能系统需求（8.2 节）。当利益相关者倾听这些观点时，软件工程必须考虑他们建立系统的语境。在众多问题中必须回答以下几个问题 [Lag10]：

- 我们能建立系统吗？
- 这些开发流程能让我们打败市场竞争对手吗？
- 有适当的资源可建立和维护假设的系统吗？
- 系统性能能满足客户需求吗？

随着时间的推移，这些问题或其他问题的答案都将随之发展变化。

8.3.2 质量功能部署

质量功能部署（Quality Function Deployment, QFD）是一种将客户要求转化成软件技术需求的技术。QFD 的“目的是最大限度地让客户从软件工程过程中感到满意” [Zul92]。为了达到这个目标，QFD 强调理解“什么是对客户有价值的”，然后在整个工程活动中部署这些价值。

在 QFD 语境中，常规需求是指在会议中向客户陈述一个产品或系统时的目标，如果这些需求存在，客户就会满意。期望需求暗指在产品或系统中客户没有清晰表述的基础功能，缺少了这些将会引起客户的不满。兴奋需求是超出客户预期的需求，当这些需求存在时会令人非常满意。

虽然 QFD 概念可应用于整个软件工程 [Par96]，但是特定的 QFD 技术可应用于需求获取活动。QFD 通过客户访谈和观察、调查以及历史数据检查（如问题报告）为需求收集活动获取原始数据。然后把这些数据翻译成需求表——称为客户意见表，并由客户和利益相关者评审。接下来使用各种图表、矩阵和评估方法抽取期望的需求并尽可能获取兴奋需求 [Aka04]。

145

关键点 QFD 以最大限度地满足客户的方式来定义需求。

建议 每个人都希望实现大量的“兴奋需求”，但是必须小心，那是“需求蔓延”开始的原因。然而另一方面，经常是兴奋需求才最终导致突破性的产品！

网络资源 有关 QFD 的更多信息请访问 www.qfdi.org。

8.3.3 使用场景

收集需求时，系统功能和特性的整体愿景开始具体化。但是在软件团队弄清楚不同类型的最终用户如何使用这些功能和特性之前，很难转移到更技术化的软件工程活动中。为实现这一点，开发人员和用户可以创建一系列的场景——场景可以识别对将要构建系统的使用线索。场景通常称为用例 [Jac92]，它描述了人们将如何使用某一系统。在 8.4 节中将更详细地讨论用例。

146

SafeHome 开发一个初步的使用场景

[场景] 一间会议室，继续首次需求收集会议。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人：我们已经讨论过通过 Internet 访问 SafeHome 功能的安全性，我想考虑得再多些。下面我们开发一个使用住宅安全功能的用户场景。

Jamie：怎么做？

主持人：我们可以采用两种不同的方法完成这个工作，但是现在，我想不要太正式吧。请问（他指向一个市场人员），你设想该如何使用这样的系统？

营销人员：嗯……好的，如果我出门在外，我想我能做的就是必须让某个没有安全码的管家或修理工进入我家。

主持人（微笑）：这就是你的理由……告诉我们实际上你怎么做？

营销人员：嗯……首先我需要一台电脑，然后登录为所有 SafeHome 用户提供的 Web 网站，提供我的用户账号……

Vinod（打断）：Web 页面必须是安全的、

加密的，以确保安全……

主持人（打断）：这是个有用信息，Vinod，但这太技术性了，我们还是只关注最终用户将如何使用该功能，好吗？

Vinod：没问题。

营销人员：那么，就像我所说的，我会登录一个 Web 网站并提供我的用户账号和两级密码。

Jamie：如果我忘记密码怎么办？

主持人（打断）：好想法，Jamie。但是我们先不谈这个。我们先把这种情况作为“异常”记录下来。一定还有其他的异常。

营销人员：在我输入密码后，屏幕将显示所有的 SafeHome 功能。我选择住宅安全功能，系统可能会要求确认我是谁，要求我的地址或电话号码或其他什么，然后显示一张图片，包括安全系统控制面板和我能执行的功能列表——安装系统、解除系统、解除一个或多个传感器。我猜还可能会允许我重新配置安全区域和其他类似的东西，但是我不确定。

（当市场营销人员继续讨论时，Doug 记录下大量内容。这些构成了最初非正式的用例场景基础。另一种方法是让市场营销人员写下场景，但这应该在会议之外进行。）

8.3.4 获取工作产品

根据将要构建的系统或产品规模的不同，需求获取后产生的工作产品也不同。对于大多数系统而言，工作产品包括：（1）要求和可行性陈述；

提问 什么样的信息是需求收集产生的？

(2) 系统或产品范围的界限说明; (3) 参与需求获取的客户、用户和其他相关利益者的名单; (4) 系统技术环境的说明; (5) 需求列表 (最好按照功能加以组织) 以及每个需求适用的领域限制; (6) 一系列使用场景, 有助于深入了解系统或产品在不同运行环境下的使用; (7) 任何能够更好地定义需求的原型。所有参与需求获取的人员需要评审以上的每一个工作产品。

147

8.3.5 敏捷需求获取

在敏捷过程中, 通过向所有利益相关者询问, 生成用户故事以获取需求。每个用户故事描述了一个从用户角度出发的简单系统需求。写在小记事卡片上的用户故事使开发者更容易选择和管理需求子集, 以便实现下一代产品的改进。敏捷倡导者主张使用用户用自己的语言编写的记事卡片, 这可以使软件开发人员的注意力转移到与利益相关者交流所选的需求, 而不是他们自己的议事日程 [Mail0a]。

关键点 在敏捷过程模型中, 用户故事是从客户中获取并记录需求的方式。

虽然用敏捷开发的方法进行需求获取吸引了很多软件团队, 但批评人士认为这种方法常常缺少对全局商业目标和非功能需求的考量。在某些情况下, 需要返工并考虑性能和安全问题。另外, 用户故事可能无法为随时间发展的系统提供足够的基础。

8.3.6 面向服务的方法

面向服务的开发将系统看作一套服务的集合。服务可能“与提供单一功能一样简单, 例如基于需求/应答机制提供一系列随机数, 或者与复杂元素整合, 例如 Web 服务 API” [Mic12]。

提问 在基于服务模型的语境中服务是什么?

面向服务开发的需求获取关注由应用系统所定义的服务。做个比喻, 想象你进入了一家高级酒店, 可享受的服务有: 门童向客人问候, 一位服务员替客人停车, 前台服务员为客人办理入住手续, 一位服务员管理着行李, 客房服务员协助客人找到所安排的房间。精心设计客人与酒店员工的联系和触点会提升客人对拜访酒店时所获得服务的印象。

许多服务设计方法强调理解客户、创造性思维和快速建立解决方案 [Mail0b], 为达到这些目标, 需求获取过程要包括人类行为研究、创新工作室和早期低精度的原型[⊖]。需求获取的技术也必须获得品牌信息和利益相关者感知的信息。另外, 为了研究如何让客户选择某个品牌, 分析师需要有策略地发现和记录新用户所渴求的质量需求。基于这一点, 用户故事将非常有帮助。

关键点 从基于服务的模型中获取的需求细化了由应用场景所衍生出的服务。每一个触点都代表用户与系统的一次交互, 从而获得所需的服务。

148

描述触点需求的特征时, 应使其与整体服务需求相呼应, 因此, 每种需求都应该可以追溯到一种特定的服务。

8.4 开发用例

在一本讨论如何编写有效用例的书中, Alistair Cockburn[Coc01b] 写道: “一个用例捕获一份‘合同’……即说明对某个利益相关者的请求做出响应时, 系统在各种条件下的行为。”

⊖ 在假设使用软件产品的环境中研究用户行为。

本质上,用例讲述了程式化的故事:最终用户(扮演多种可能角色中的一个)如何在特定环境下和系统交互。这个故事可以是叙述性的文本、任务或交互的概要、基于模板的说明或图形表示。不管其形式如何,用例都从最终用户的角度描述了软件或系统。

撰写用例的第一步是确定故事中所包含的“参与者”。参与者是在将要说明的功能和行为环境内使用系统或产品的各类人员(或设备)。参与者代表了系统运行时人(或设备)所扮演的角色,更为正式的定义是:参与者是任何与系统或产品通信的事物,且对系统本身而言参与者是外部的。在使用系统时,每个参与者都有一个或多个目标。

要注意的是,参与者和最终用户并非一回事。典型的用户可能在使用系统时扮演了许多不同的角色,而参与者表示了一类外部实体(经常是人员,但并不总是如此),在用例中他们仅扮演一种角色。例如机床操作员(一个用户),他和生产车间(其中布置了许多机器人和数控机床)内的某个控制计算机交互。在仔细考察需求后,控制计算机的软件需要4种不同的交互模式(角色):编程模式、测试模式、监控模式和故障检查模式。因此,4个参与者可定义为:程序员、测试员、监控员和故障检修员。有些情况下,机床操作员可以扮演所有这些角色,而另一些情况下,每个参与者的角色可能由不同的人员扮演。

需求获取是一个逐步演化的活动,因此在第一次迭代中并不能确认所有的参与者。在第一次迭代中有可能识别主要的参与者 [Jac92],而对系统了解更多之后,才能识别出次要的参与者。主要参与者直接且经常使用软件,他们要获取所需的系统功能并从系统得到预期收益。次要参与者为系统提供支持,以便主要参与者能够完成他们的工作。

一旦确认了参与者,就可以开发用例了。对于应该由用例回答的问题, Jacobson [Jac92] 提出了以下建议:^①

- 主要参与者和次要参与者分别是谁?
- 参与者的目标是什么?
- 故事开始前有什么前提条件?
- 参与者完成的主要工作或功能是什么?
- 按照故事所描述的还可能需要考虑什么异常?
- 参与者的交互中有什么可能的变化?
- 参与者将获得、产生或改变哪些系统信息?
- 参与者必须通知系统外部环境的改变吗?
- 参与者希望从系统获取什么信息?
- 参与者希望得知意料之外的变更吗?

回顾基本的 SafeHome 需求,我们定义了4个参与者:房主(用户)、配置管理人员(很可能就是房主,但扮演不同的角色)、传感器(附属于系统的设备)和监控子系统(监控 SafeHome 房间安全功能的中央站)。仅从该例子的目的来看,我们只考虑了房主这个参与者。房主通过使用报警控制面板或计算机等多种方式和住宅安全功能交互:(1)输入密码以便能进行其他交互;(2)查询安全区的状态;(3)查询传感器的状态;(4)在紧急情况时按

关键点 用例是从参与者的角度定义的。参与者是人员(用户)或设备在和软件交互时所扮演的角色。

网络资源 一篇非常好的关于用例的论文可以从 www.ibm.com/developerworks/web_services/library/co-design7.html 下载。

149

提问 为了开发有效的用例我需要什么知道什么?

① Jacobson 的问题已经被扩展到为用例场景提供更复杂的视图。

下应急按钮；(5) 激活或关闭安全系统。

考虑房主使用控制面板的情况，系统激活的基本用例如下。[⊖]

- 1. 房主观察 SafeHome 控制面板 (图 8-1)，以确定系统是否已准备好接收输入。如果系统尚未就绪，“not ready”消息将显示在 LCD 显示器上，房主必须亲自动手关闭窗户或门以使得“not ready”消息消失。(not ready 消息意味着某个传感器是开着的，即某个门或窗户是开着的。)
- 2. 房主使用键盘键入 4 位密码，系统将该密码与已存储的有效密码相比较，如果密码不正确，控制面板将鸣叫一声并自动复位以等待再次输入，如果密码正确，控制面板将等待进一步的操作。
- 3. 房主选择键入“stay”或“away”(图 8-1)以启动系统。“stay”只激活外部传感器(内部的运动监控传感器是关闭的)，“away”激活所有的传感器。
- 4. 激活时，房主可以看到一个红色的警报灯。

150

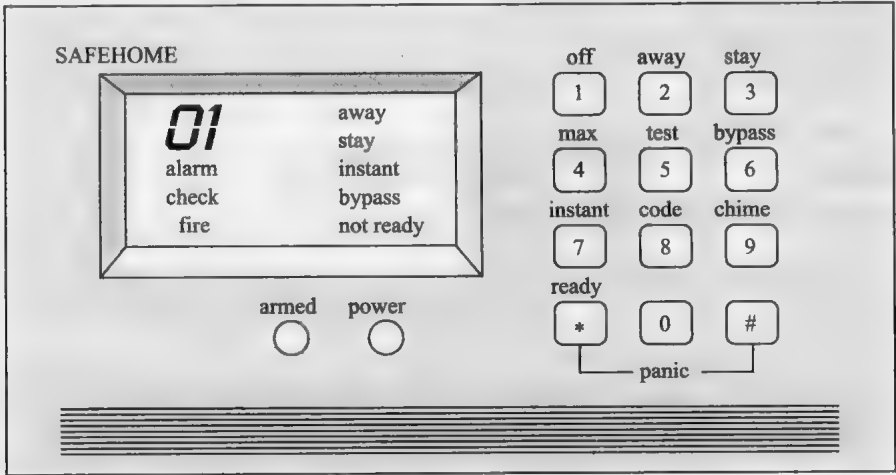


图 8-1 SafeHome 控制面板

基本用例从较高层次上给出参与者和系统之间交互的故事。

在很多情况下，需要进一步细化用例以便为交互提供更详细的说明。

例如，Cockburn[Coc01b] 建议使用如下模板详细说明用例。

用例：初始化监控。

主要参与者：房主。

目标：在房主离开住宅或留在房间时，设置系统以监控传感器。

前提条件：系统支持密码输入和传感器识别功能。

触发器：房主决定“设置”系统，即打开警报功能。

场景：

- 1. 房主：观察控制面板。
- 2. 房主：输入密码。
- 3. 房主：选择“stay”或“away”。

建议 用例通常写得不规范，但是使用这里介绍的模板可以确保你已经说明了所有关键问题。

151

⊖ 注意，该用例和通过 Internet 访问系统的情形不同，该用例的环境是通过控制面板交互而不是使用计算机或移动设备所提供的图形用户接口 (GUI)。

4. 房主：观察红色报警灯显示 SafeHome 已经被打开。

异常：

1. 控制面板没有准备就绪：房主检查所有的传感器，确定哪些是开着的（即门窗是开着的），并将其关闭。
2. 密码不正确（控制面板鸣叫一声）：房主重新输入正确的密码。
3. 密码不识别：必须对监控和响应子系统重新设置密码。
4. 选择 stay：控制面板鸣叫两声并且 stay 灯点亮；激活边界传感器。
5. 选择 away：控制面板鸣叫三声并且 away 灯点亮；激活所有传感器。

优先级：必须实现。

何时可用：第一个增量。

使用频率：每天多次。

使用方式：通过控制面板接口。

次要参与者：技术支持人员，传感器。

次要参与者使用方式：

技术支持人员：电话线。

传感器：有线或无线接口。

未解决的问题：

1. 是否还应该有不使用密码或使用缩略密码激活系统的方式？
2. 控制面板是否还应显示附加的文字信息？
3. 房主输入密码时，从按下第一个按键开始必须在多长时间内输入密码？
4. 在系统真正激活之前有没有办法关闭系统？

可以使用类似的方法开发其他房主的交互用例。重要的是必须认真评审每个用例。如果某些交互元素模糊不清，用例评审将解决这些问题。

[152]

SafeHome 开发高级用例图

【场景】会议室，继续需求收集会议。

【人物】Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

【对话】

主持人：我们已经花费了相当多的时间讨论 SafeHome 住宅安全功能。在休息时我画了一个用例图，用它来概括重要的场景，这些场景是该功能的一部分。大家看一下。

（所有的与会者注视图 8-2。）

Jamie：我恰好刚开始学习 UML 符号^①。住宅安全功能是由中间包含若干椭圆的大方框表示吗？而且这些椭圆代表我们已经用文字写下的用例，对吗？

主持人：是的。而且棍型小人代表参与者——与系统交互的人或事物，如同用例中所描述的……哦，我使用作了标记的矩形表示在这个用例中那些不是人而是传感器的参与者。

Doug：这在 UML 中合法吗？

① 不熟悉 UML 符号的读者请参考附录 1 中的 UML 基本指南。

主持人：合法性不是问题，重点是交流信息。我认为使用棍型小人代表设备可能会产生误导，因此我做了一些改变。我认为这不会产生什么问题。

Vinod：好的。这样我们就为每个椭圆进行了用例说明，还需要生成更详细的基于模板的说明吗？我们已经阅读过那些说明了。

主持人：有可能，但这可以等到考虑完其他的 SafeHome 功能之后。

营销人员：等一下，我已经看过这幅图，突然间我意识到我们遗漏了什么。

主持人：哦，是吗。告诉我们遗漏了什么。（会议继续进行。）

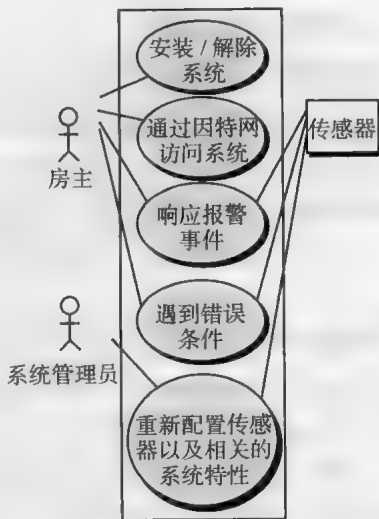


图 8-2 SafeHome 住宅安全系统功能的 UML 用例图

153

软件工具 用例开发

【目标】 通过使用能提高访问透明性和一致性的自动化模板和机制来协助开发用例。

【机制】 工具的原理各不相同。通常，用例工具为创建有效用例提供填空式的模板。大多数用例功能能嵌入一系列更宽泛的需求工程功能中。

【代表性工具】^①

大量的分析建模工具（多数基于 UML）可为用例开发和建模提供文字和图形化支持。

- Objects by Design. UML 工具资源 (www.objectsbydesign.com/tools/umltools_byCompany.html), 提供对该类工具的全面链接。

8.5 构建分析模型^②

分析模型的作用是为基于计算机的系统提供必要的信息、功能和行为域的说明。随着软件工程师更多地了解将要实现的系统以及其他相关利益者更多地了解他们到底需要什么，模型应能够动态变更。因此，分析模型是任意给定时刻的需求快照，我们对这种变更应有思想准备。

随着分析模型的演化，某些元素将变得相对稳定，为后续设计任务提供稳固的基础。但是，有些模型元素可能是不稳定的，这表明利益相关者仍然没有完全理解系统的需求。分析模型及其构建方法将在第 9 ~ 11 章详细说明，下面仅提供简要的概述。

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名被各自的开发者注册为商标。

② 在本书中我把分析模型和需求模型作为同义词使用，它们都用于描述信息、功能和行为领域的问题需求。

8.5.1 分析模型的元素

有很多不同的方法可用来考察计算机系统的需求。某些软件人员坚持最好选择一个表达模式（例如用例）并排斥所有其他的模式。有些专业人士则相信使用许多不同的表达模式来描述分析模型是值得的，不同的表达模式促使软件团队从不同的角度考虑需求——一种方法更有可能造成需求遗漏、不一致性和歧义性。一些普遍的元素对大多数分析模型来说都是通用的。

建议 把利益相关者包括进来通常是个好主意。做到这一点最好的方法之一是让每个利益相关者写下描述将如何使用软件的用例。

基于场景的元素。使用基于场景的方法可以从用户的视角描述系统。例如，基本的用例（8-4 节）及其相应的用例图（图 8-2）可演化成更精细的基于模板的用例。需求模型的基于场景的元素通常是正在开发的模型的第一部分。同样，它们也作为创建其他建模元素时的输入。例如，图 8-3 是获取需求并用用例进行表述的 UML 活动图^①，图中给出了最终基于场景的三层详细表达。

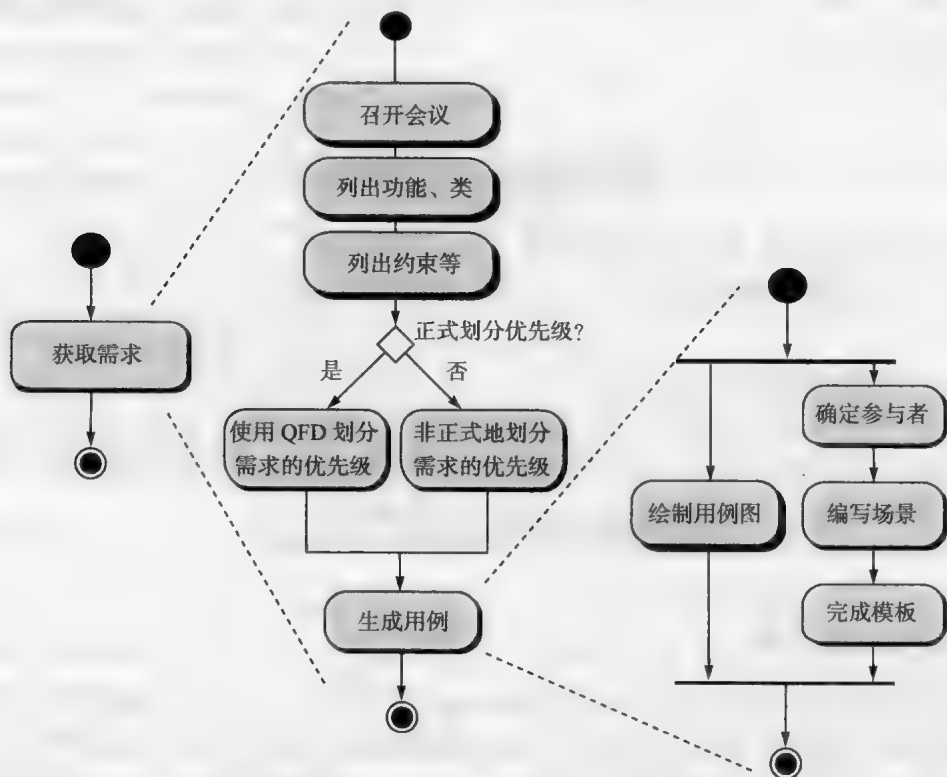


图 8-3 获取需求的 UML 活动图

基于类的元素。每个使用场景都意味着当一个参与者和系统交互时所操作的一组对象，这些对象被分成类——具有相似属性和共同行为的事物集合。例如，可以用 UML 类图描绘 SafeHome 安全功能的 Sensor 类，如图 8-4 所示。注意，UML 类图列出了传感器的属性（如 name、type）和可以用于修改这些属性的操作（如 identify、enable）。除了类图，其他分析建模元素描绘了类之间的协作以及类之间的关联和交互。在第 10 章中将

建议 一种分离类的方法是查找用例脚本中的叙述性名词。至少某些名词将是候选类。第 12 章中将详细说明这一点。

① 不熟悉 UML 符号的读者请参考附录 1 中的 UML 基本指南。

有更详细的讨论。

行为元素。基于计算机的系统行为能够对所选择的设计和所采用的实现方法产生深远的影响。因此，需求分析模型必须提供描述行为的建模元素。

状态图是一种表现系统行为的方法，该方法描绘系统状态以及导致系统改变状态的事件。状态是任何可以观察到的行为模式。另外，状态图还指明了在某个特殊事件后采取什么动作（例如激活处理）。

为了更好地说明状态图的使用，考虑将软件嵌入 SafeHome 的控制面板，并负责读取用户的输入信息。简化的 UML 状态图如图 8-5 所示。

关键点 状态是外部可观察到的行为模式，外部激励导致状态间的转换。

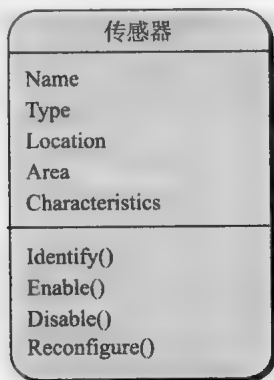


图 8-4 Sensor 类图

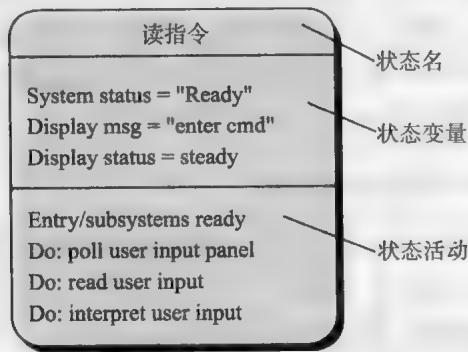


图 8-5 UML 状态图表示

另外，作为一个整体的系统行为表述也能够建模于各个类的行为之上。第 11 章将有更多关于行为建模的讨论。

156

SafeHome 初步的行为建模

[场景] 会议室，继续需求会议。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人：我们刚才差不多已经讨论完了 SafeHome 的住宅安全功能。但是在结束之前，我希望讨论一下功能的行为。

营销人员：我不太理解你所说的行为意味着什么。

Ed（大笑）：那就是如果产品行为错误就让它“暂停”。

主持人：不太准确，让我解释一下。

（主持人向需求收集团队解释行为建模的

基本知识。）

营销人员：这看起来有点技术性，我不敢确定能不能在这里帮上忙。

主持人：你当然可以。你从用户的角度观察到什么行为？

营销人员：嗯……好的，系统将监控传感器、从房主那里读指令，还将显示其状态。

主持人：看到了吧，你是可以帮上忙的。

Jamie：还应该使用计算机确定是否有任何输入，例如基于 Internet 的访问或配置信息。

Vinod：是的，实际上，配置系统是其权利内的一个状态。

Doug：你这家伙开始转过弯儿了，让我们多想一些……有方法把这个画出来吗？

主持人：有方法，但等到会后再开始吧。

8.5.2 分析模式

任何有一些软件项目需求工程经验的人都开始注意到,在特定的应用领域内某些事情在所有的项目中重复发生^①。这些分析模式 [Fow97] 在特定应用领域内提供一些解决方案(如类、功能、行为),在为许多应用项目建模时都可以重复使用。

Geyer-Schulz 和 Hahsler[Gey01] 提出了使用分析模式的两个优点:

首先,分析模式提高了抽象分析模型的开发速度,通过提供可重复使用的分析模型捕获具体问题的主要需求,例如关于优点和约束的说明。其次,通过建议的设计模式和可靠的通用问题解决方案,分析模式有利于把分析模型转化为设计模型。

建议 如果你想更快获得用户需求并为你的团队提供已经证实可用的方法,那就使用分析模式。

[157]

通过参照模式名称可把分析模式整合到分析模型中。同时,这些分析模式还将存储在仓库中,以便需求工程师能通过搜索工具发现并应用它们。在标准模板 [Gey01]^② 中会提供关于分析模式(和其他类型模式)的信息,更多细节在第 16 章讨论。分析模式的样例和有关这一论题更多的讨论在第 11 章中。

8.5.3 敏捷需求工程

敏捷需求工程的意图是把利益相关者的思想传递给软件团队,而不是生成扩展的分析工作产品。在许多情况下,需求未被预定义,但可作为每次产品迭代开发的开始。当敏捷团队深入地理解了产品的关键特性时,与下一个产品的增量相关的用户故事(第 5 章)便可得到精炼。敏捷过程鼓励尽早定义和实施优先级最高的产品特性,这样能尽早生成并测试工作原型。

敏捷需求工程涉及软件项目中一些常见的重要问题:需求高发散性,不完整的开发技术知识,客户在看到工作原型之前不能清晰表达他们的愿景。敏捷过程将需求过程和设计活动分离开来。

8.5.4 自适应系统的需求

自适应系统^③能自我调整配置、增加功能、自我保护并从失效中恢复,而且,在完成这些活动时,其内部复杂性是对用户隐藏的 [Qur09]。自适应需求阐明了自适应系统的各种必备的变化性。当在同一时间指定软件产品的一种功能或质量表现时,这意味着需求中必定包含着变化性或灵活性的概念。变化性可能包括时间的不确定性、用户角色的差别(例如最终用户与系统管理员的差别)、基于问题域的行为扩展(例如商业或教育)或利用系统资产预定义行为。

提问 自适应系统的特点是什么?

捕获自适应需求时所关注的问题与传统系统中需求工程所关注的问题相同。然而,在逐一回答这些问题时,可将其表示成特定的变化性。答案变化越大,结果系统的复杂性越大,这样才能容纳这些需求。

[158]

① 在某些情况下,事情会重复发生而不论应用领域是什么。例如,不管所考虑的应用领域是什么,用户接口的特点和功能都是共有的。

② 文献中已经提出了各种各样的模式模板,感兴趣的读者可以参阅 [Fow97]、[Gam95]、[Yac03] 和 [Bus07]。

③ 自适应系统的一个实例是“位置警告”应用,它根据所在移动平台的位置来自适应地调整系统的行为。

8.6 协商需求

在一个理想的需求工程情境中,起始、获取和细化工作能确保得到足够详细的客户需求,以开展后续的软件工程步骤。但遗憾的是,这几乎不可能发生。实际上,一个或多个利益相关者恐怕得进入到协商的过程中,在多数情况下要让利益相关者以成本和产品投放市场的时间为背景,平衡功能、性能和其他的产品或系统特性。这个协调过程的目的是保证所开发的项目计划,在满足利益相关者要求的同时反映软件团队所处真实世界的限制(如时间、人员、预算)。

最好的协商是争取“双赢”的结果^①。利益相关者的“赢”在于获得能满足客户大多数需要的系统或产品;而作为软件团队一员,“赢”在于按照实际情况、在可实现的预算和时间期限内完成工作。

Boehm[Boe98]定义了每个软件过程迭代启动时的一系列协商活动。不是定义单一的客户交流活动,而是定义了如下系列活动:

1. 识别系统或子系统中关键的利益相关者。
2. 确认利益相关者“赢”的条件。
3. 就利益相关者“赢”的条件进行协商,以便使其与所有相关的(包括软件团队)一系列双赢条件一致。

这些初始步骤的成功实施可以达到双赢的结果,这是继续开展后续软件工程活动的关键。

引述 折衷是这样一种艺术:采用某种方式分蛋糕,让每个人感觉自己得到了最大的一块。

Ludwig Erhard

网络资源 可以从以下地址下载一篇关于软件需求协商的简短文章: www.alexanderegyed.com/publications/Software_Requirements_Negotiation-Some_Lessons_Learned.html。

159

信息栏 协商的艺术

学习如何有效地协商可以帮助你更好地度过个人生活或技术生涯。如下指导原则非常值得考虑:

1. 认识到这不是竞争。为了成功,为了获得双赢,双方将不得不妥协。
2. 制定策略。判定你希望得到什么、对方希望得到什么,以及你将如何行动以使这两方面的希望都能实现。
3. 主动倾听。不要在对方正在说话时做出程式化的响应。听是为了获取信息,这些信息有助于在磋商中更好地说明你的

立场。

4. 关注对方的兴趣。如果想避开冲突,就不要太过于坚持自己的立场。
5. 不要进行人身攻击。应集中关注需要解决的问题。
6. 要有创新性。处于僵局时不要害怕,而应考虑如何摆脱困境。
7. 随时准备做出承诺。一旦已经达成一致,不要闲聊胡扯,马上做出承诺然后继续进行。

Fricker[Fri10]和他的同事建议不再采用传统的需求规格说明书方式,而是代之以称作为握手的双向沟通过程。在握手过程中,软件团队提出需求解决方案,描述它们的影响,与客

① 有许多关于谈判技巧的书籍(如[Fis11]、[Lew09]、[Rai06])。这是一个应该学习的许多重要技巧之一。读一本吧。

户代表沟通他们的意图，客户代表审核提议的解决方案，关注丢失的特性并寻求新需求的清晰性。如果客户接受提议的解决方案，则说明需求是足够好的。

握手允许把详细需求托付给软件团队。团队需要从客户（例如产品用户和领域专家）中获取需求，从而提高产品的接受度。握手方法有助于多样需求的识别、分析和多样选择，并可促进双赢的协商。

SafeHome 开始协商

[场景] Lisa Perez 的办公室，在第一次需求收集会议之后。

[人物] Doug Miller，软件工程经理；Lisa Perez，市场营销经理。

[对话]

Lisa: 我听说第一次会议进行得很好。

Doug: 确实是这样，你派了几个有经验的人参加会议……他们确实有很多贡献。

Lisa (微笑): 是的，他们的确告诉我他们融入了会议，而且会议卓有成效。

Doug (大笑): 下次再见面时我一定要脱帽致敬……看，Lisa，我想在你们主管所说的日期内获取所有的住宅安全功能可能会有困难。我知道现在还早，但是我们已经有一些落后于原定计划，并且……

Lisa (皱眉): 我们必须在那个时间获得产品，Doug。你说的是什么功能？

Doug: 我认为我们可以在截止日期前完成所有的住宅安全功能，但是必须把 Internet 访问功能推迟到第二次发布的产

品中加以考虑。

Lisa: Doug，Internet 访问是 SafeHome 最引人注目之处，我们正在围绕这一点开发我们整体的营销活动。我们必须实现它！

Doug: 我理解你的处境，我确实理解。问题在于为了向你提供 Internet 访问，我们将需要一整套 Web 站点安全防护措施，这将花费时间和人力。我们还必须在首次发布的产品中开发很多的附加功能……我认为我们不能在现有资源下完成这些。

Lisa (皱眉): 我知道，但你必须找到实现方法，Internet 访问对住宅安全功能非常关键，对其他功能也很关键……其他功能可以等到下一次发布的产品再予以考虑……我同意这样。

很明显 Lisa 和 Doug 陷入了僵局，而且他们必须协商出一个解决办法。他们能够“双赢”吗？如果你扮演调解人的角色，有什么建议？

8.7 需求监控

当今常见的软件项目是增量开发项目。这意味着需要扩展用例，针对每次新的软件增量开发新的测试用例，并贯穿整个项目进行源代码的不断集成。在实现增量开发时，需求监控显得尤为有益。其中包括 5 项任务：（1）分布式调试以发现错误并找到出错的原因；（2）进行运行验证，确认软件与规格说明是否匹配；（3）进行运行确认以评估逐步扩展的软件是否满足用户目标；（4）实施商业活动监控以评估系统是否满足商业目标；（5）演化与协同设计可为那些作为系统演化者的利益相关者提供信息。

增量开发意味着增量确认的要求。需求监控支持持续确认，通过分析用户目标模型以使

软件的逐步改进 [Rob10]。

8.8 确认需求

当需求模型的每个元素都已完成创建后，需要检查一致性、是否有遗漏以及是否有歧义性。模型所表现的需求由利益相关者划分优先级并组合成一个整体，该需求整体将以软件增量形式逐步实现。需求模型的评审将提出如下问题：

- 每项需求都与系统或产品的整体目标一致吗？
- 所有的需求都已经在相应的抽象层上说明了吗？换句话说，是否有一些需求是在技术细节过多的层次上提出的，并不适合当前的阶段？
- 需求是真正必需的，还是另外加上去的，有可能不是系统目标所必需的特性吗？
- 每项需求都有界定且无歧义吗？
- 每项需求都有归属吗？换句话说，是否每项需求都标记了来源（通常是一个明确的人）？
- 有需求和其他需求相冲突吗？
- 在系统或产品所处的技术环境下每个需求都能够实现吗？
- 一旦实现后，每个需求是可测试的吗？
- 需求模型恰当地反映了将要构建系统的信息、功能和行为吗？
- 需求模型是否已经使用合适的方式“分割”，能够逐步地揭示详细的系统信息？
- 已经使用需求模式简化需求模型吗？已经恰当地确认了所有的模式吗？所有的模式都与客户的需求一致吗？

提问 评审需求时，我将提出什么问题？

161

应当提出以上这些问题和其他一些问题，并回答问题，以确保需求模型精确地反映利益相关者的需求并为设计奠定坚实的基础。

8.9 避免常见错误

Buschmann[Bus10] 描述了作为软件团队实施需求工程时必须避免的三个相关错误，即对特性、灵活性和性能的过分偏好。

特性偏好是指以功能覆盖率来表征整体系统质量的做法。某些组织倾向于尽可能提早交付等量的功能，从而保证最终产品的整体质量。认为越多越好的商务利益相关者会参与驱动这些事。还有一种软件开发者倾向快速实施简易功能，而不考虑它们的质量。事实上软件项目失败的最常见原因之一是缺少可实用的质量——而不是丢失功能。为了避免落入这个陷阱，你应与其他利益相关者讨论系统必需的关键功能，确保每项已交付的功能都具备了所有必要的质量特性。

灵活性偏好发生在软件工程师过分重视产品的自适应性和配置便利性时。过分灵活的系统会很难配置，并且可操作性差，这是系统范围定义混乱的征兆。然而根本原因可能是开发者使用灵活性来应对不确定性，而不是尽早定稿设计方案。他们提供设计“钩子”，从而允许增加计划外的特性。结果是“灵活”系统产生了不必要的复杂性，越难测试就会有越多的管理挑战。

性能偏好是指软件开发者过分关注质量特性的方面的系统性能开销，如可维护性、可靠性和安全性。系统性能特性应该部分取决于非功能软件需求的评估。性能应该与产品的商业需求一致，同时必须与其他系统特性相兼容。

8.10 小结

162

需求工程的任务是为设计和构建活动建立一个可靠且坚固的基础。需求工程发生在为通用软件过程定义的沟通活动和建模活动中。软件团队成员要完成 7 个不同的需求工程任务：起始、获取、细化、协商、规格说明、确认和管理。

在项目起始阶段，利益相关者建立基本的问题需求，定义最重要的项目约束并陈述主要的特性和功能，必须让系统表现出这些特性和功能以满足其目标。该信息在获取阶段得到提炼和延伸，在此阶段中利用有主持人的会议、QFD 和使用场景的开发进行需求收集活动。

细化阶段进一步把需求扩展为分析模型——基于场景、基于活动、基于类、行为和面向数据流的模型元素集合。模型可以参考分析模式——已经在不同应用系统中重复出现的问题域特征。

在确定需求和创建分析模型时，软件团队和其他利益相关者协商优先级、可用性和每项需求的相对成本。协商的目标是制定一个现实可行的项目计划。此外，将按照客户需求确认每项需求和整个需求模型，以确认将要构建的系统对于客户的要求是正确的。

习题与思考题

- 8.1 为什么大量的软件开发人员没有足够重视需求工程？以前有没有什么情况让你可以跳过需求工程？
- 8.2 你负责从一个客户处获取需求，而他告诉你太忙了没时间见面，这时你该怎么做？
- 8.3 讨论一下当需求必须从三四个不同的客户中提取时会发生什么问题。
- 8.4 为什么我们说需求模型表现了系统的时间快照？
- 8.5 让我们设想你已经说服客户（你是一个绝好的销售人员）同意你作为一个开发人员所提出来的每一个要求，这能够让你成为一个高明的协商人员吗？为什么？
- 8.6 想出 3 个以上在需求起始阶段可能要问利益相关者的“与环境无关的问题”。
- 8.7 开发一个促进需求收集的“工具包”。工具包应包含：一系列需求收集会议的指导原则，用于协助创建列表的材料，以及其他任何可能有助于定义需求的条款。
- 8.8 你的指导老师将把班级分成 4 或 6 人的小组，组中一半的同学扮演市场部的角色，另一半将扮演软件工程部的角色。你的工作是定义本章所介绍的 SafeHome 安全功能的需求，并使用本章所提出的指导原则引导需求收集会议。
- 8.9 为如下活动之一开发一个完整的用例：
 - a. 在 ATM 提款。
 - b. 在餐厅使用信用卡付费。
 - c. 使用一个在线经纪人账户购买股票。
 - d. 使用在线书店搜索书（某个指定主题）。
 - e. 你的指导老师指定的一个活动。
- 8.10 用例“异常”代表什么？
- 8.11 选取一个问题 8.9 中列举的活动，写一个用户故事。
- 8.12 考虑问题 8.9 中你生成的用户用例，为应用系统写一个非功能性需求。
- 8.13 用你自己的话描述一个分析模式。

163

8.14 使用 8.5.2 节描述的模板, 为下列应用领域建议一个或多个分析模式:

- a. 会计软件
- b. E-mail 软件
- c. 互联网浏览器
- d. 字符处理系统
- e. 网站生成系统
- f. 由指导老师特别指定的应用领域

8.15 在需求工程活动的协商情境中, “双赢”意味着什么?

8.16 你认为当需求确认揭示了一个错误时将发生什么? 谁将参与错误修正?

8.17 哪 5 个任务组成了综合需求监控程序?

扩展阅读与信息资源

因为需求工程是成功创建任何复杂的基于计算机的系统的核心, 所以大量的书籍都在讨论需求工程。Chemuturi (《Requirements Engineering and Management of Software Development Projects》, Springer, 2013) 阐明了需求工程的重要部分, Pohl 和 Rupp (《Requirements Engineering Fundamentals》, Rocky Nook, 2011) 表述了基本原理和观念, Pohl (《Requirements Engineering》, Springer, 2010) 提供了一个完整的需求工程流程的详细视图。Young (《The Requirements Engineering Handbook》, Artech House Publishers, 2003) 对需求工程任务进行了更高层次的讨论。

Beaty 和 Chen (《Visual Models for Software Products Best Practices》, Microsoft Press, 2012), Robertson (《Mastering the requirements Process: Getting Requirements Right》, 3rd ed., Addison-Wesley, 2012), Hull 和她的同事 (《Requirements Engineering》, 3rd ed., Springer-Verlag, 2010), Bray (《An Introduction to Requirements Engineering》, Addison-Wesley, 2002), Arlow (《Requirements Engineering》, Addison-Wesley, 2001), Gilb (《Requirements Engineering》, Addison-Wesley, 2000), Graham (《Requirements Engineering and Rapid Development》, Addison-Wesley, 1999), Sommerville 和 Kotonya (《Requirement Engineering: Processes and Techniques》, Wiley, 1998), 等等, 他们都讨论了需求工程这一主题。Wiegiers (《More About Software Requirements》, Microsoft Press, 2010) 提供了很多需求收集和管理的技术实践。

Withall (《Software Requirement Patterns》, Microsoft Press, 2007) 描述了基于模式视图的需求工程。Ploesch (《Contracts, Scenarios and Prototypes》, Springer-Verlag, 2004) 讨论了开发软件需求的先进技术。Windle 和 Abreo (《Software Requirements Using the Unified Process》, Prentice-Hall, 2002) 从统一过程和 UML 符号的角度讨论了需求工程。Alexander 和 Steven (《Writing Better Requirements》, Addison-Wesley, 2002) 提出了一套简短的指导原则, 目的是编写清楚的需求, 使用场景表现需求并评审最终结果。

用例建模通常在创建分析模型的所有其他方面时使用, 讨论该主题的资料有: Rosenberg 和 Stephens (《Use Case Driven Object Modeling with UML: Theory and Practice》, Apress, 2007), Denny (《Succeeding with Use Cases: Working Smart to Deliver Quality》, Addison-Wesley, 2005), Alexander 和 Maiden (eds.) (《Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle》, Wiley, 2004)。Leffingwell 和他的同事 (《Managing Software Requirements: A Use Case Approach》, 2nd ed., Addison-Wesley, 2003) 表述了非常有用的需求收集的最佳实践。

有些书讨论了敏捷需求, 包括: Adzic (《Specification by Example: How Successful Teams Deliver

the Right Software 》, Manning Publications, 2011), Leffingwell (《 Agile Requirements: Lean Requirements for Teams, Programs, and Enterprises 》, Addison-Wesley, 2011), Cockburn (《 Agile Software Development: The Cooperative Game 》, 2nd ed., Addison-Wesley, 2006), Cohn (《 User Stories Applied: For Agile Software Development 》, Addison -Wesley, 2004)。

网上有大量丰富的关于需求工程和分析的信息资源。与需求工程和分析相关的最新参考文献可以在 SEPA 网站 <http://www.mhhe.com/pressman> 上找到。

需求建模：基于场景的方法

要点浏览

概念：文字记录是极好的交流工具，但并不一定是表达计算机软件需求的最好方式。需求建模使用文字和图表的复合形式，以相对容易理解的方式描绘需求，更重要的是，可以更直接地评审它们的正确性、完整性和一致性。

人员：软件工程师（有时被称作分析师）使用从客户那里获取的需求来构建模型。

重要性：为了确认软件需求，你需要从不同的视角检验需求。本章将从基于场景的观点考虑需求建模并检验如何在UML中使用补充场景。在第10章和第11章会学习需求建模的其他“维度”。在检验大量不同维度时，会增加发现错误的概率。这些错误可能是与表象不一致，或可能是没有发现的缺失项。

步骤：基于场景的建模从用户的角度表现系统。在基于场景建模时，将更好地理解用户如何与软件交互，发现没有覆盖到的利益相关者所需的主要系统功能和特性。

工作产品：基于场景建模产生的面向文本的表达称作“用例”。用例描述了特定交互方式，形成非正式（只简单描述）或更加结构化或正规化的自然特征。这些用例能补充大量不同的UML图，覆盖更多交互的程序化观点。

质量保证措施：必须评审需求建模工作产品的正确性、完整性和一致性，必须反映所有利益相关者的要求并为从中导出设计建立基础。

在技术层面上，软件工程开始于一系列的建模工作，最终生成待开发软件的需求规格说明和设计表示。需求模型实际上是一组模型^①，是系统的第一个技术表示。

在一本关于需求建模方法的开创性书籍中，Tom DeMarco[Dem79]如下这样描述该过程：

回顾分析阶段的问题和过失，我建议对分析阶段的目标进行以下的增补。分析的结果必须是高度可维护的，尤其是要将此结果应用于目标文档（软件需求规格说明）。必须使用一种有效的分割方法解决规模问题，维多利亚时代小说式的规格说明是不行的。尽可能使用图形符号。考虑问题时必须区分逻辑的（本质）和物理的（实现）……无论如何，我们至少需要……某种帮助我们划分需求的方法，并在规格说明前用文档记录该划分……某种跟踪和评

关键概念

活动图
域分析
正式用例
需求分析
需求建模
基于场景建模
泳道图
UML 模型
用例
用例异常

① 本书过去的版本使用分析模型这个术语而不是需求模型。本版中决定使用这两个术语，以便表达在解决问题的不同方面时定义的建模活动。分析是获取需求时的动作。

估接口的手段……使用比叙述性文本更好的新工具来描述逻辑和策略……

尽管 DeMarco 在 25 年前就写下了关于分析建模的特点，但他的意见仍然适用于现代的需求建模方法和表示方法。

9.1 需求分析

需求分析产生软件工作特征的规格说明，指明软件和其他系统元素的接口，规定软件必须满足的约束。在需求分析过程中，软件工程师（有时这个角色也被称作分析师或建模师）可以细化在前期需求工程的起始、获取、协商任务中建立的基础需求（第 8 章）。

需求建模动作结果为以下一种或多种模型类型：

- 场景模型：出自各种系统“参与者”观点的需求。
- 面向类的模型：表示面向对象类（属性和操作）的模型，其方式为通过类的协作获得系统需求。
- 基于行为和模式的模型：描述如何将软件行为看作外部“事件”后续的模型。
- 数据模型：描述问题信息域的模型。
- 面向流的模型：表示系统的功能元素并且描述当功能元素在系统中运行时怎样进行数据变换。

这些模型为软件设计者提供信息，这些信息可以被转化为结构、接口和构件级的设计。最终，在软件开发完成后，需求模型（和需求规格说明）就为开发人员和客户提供了评估软件质量的手段。

本章关注基于场景的建模，这项技术在整个软件工程界发展迅猛。在第 10 章和第 11 章，我们考虑基于类的模型和行为模型。在过去十几年，人们已经不常使用流和数据建模，而逐步流行使用场景和基于类的方法，以此作为行为方法和基于模式技术的补充。^①

9.1.1 总体目标和原理

在整个分析建模过程中，软件工程师的主要关注点集中在做什么而不是怎么做。在特定环境下发生哪些用户交互？系统处理什么对象？系统必须执行什么功能？系统展示什么行为？定义什么接口？有什么约束？^②

在前面的章节中，我们注意到在该阶段要得到完整的需求规格说明是不可能的。客户也许无法精确地确定想要什么，开发人员也许无法确定能恰当地实现功能和性能的特定方法，这些现实情况都削弱了迭代需求分析和建模方法的效果。分析师将为已经知道的内容建模，并使用该模型作为软件进一步扩展的设计基础。^③

需求模型必须实现三个主要目标：（1）描述客户需要什么；（2）为软件设计奠定基础；

引述 任何一个需求“视图”都不足以理解和描述一个复杂系统所需的行为。

Alan M. Davis

关键点 一旦软件完成后，分析模型和需求规格说明书将成为评估软件质量的手段。

引述 需求不是架构。需求既不是设计，也不是用户接口。需求就是指需要什么。

Andrew Hunt,
David Thomas

① 在这一版本中我们省略了面向流建模和数据建模。但是在网上可以找到这些较老的需求建模方法的大量信息。如果你感兴趣，可以搜索关键词“结构化分析”进行查找。

② 应该注意，当客户变得更加精通技术时，规格说明书中的“怎么做”需同“做什么”一样重要。但是，基本关注点应保留在“做什么”上。

③ 软件团队也可以花些功夫选择生成一个原型（第 4 章），以便更好地理解系统的需求。

(3) 定义在软件完成后可以被确认的一组需求。分析模型在系统级描述和软件设计(第12~18章)之间建立了桥梁。这里的系统级描述给出了在软件、硬件、数据、人员和其他系统元素共同作用下的整个系统或商业功能,而软件设计给出了软件的应用程序结构、用户接口以及构件级的结构。这个关系如图9-1所示。

重要的是要注意需求模型的所有元素都可以直接跟踪到设计模型。通常难以清楚地区分这两个重要的建模活动之间的设计和分析工作,有些设计总是作为分析的一部分进行,而有些分析将在设计中进行。

9.1.2 分析的经验原则

Arlow 和 Neustadt[Arl02] 提出了大量有价值的经验原则,在创建分析模型时应该遵循这些经验原则:

- 模型应关注在问题域或业务域内可见的需求,抽象的级别应该相对高一些。“不要陷入细节”[Arl02],即不要试图解释系统将如何工作。
- 需求模型的每个元素都应能增加对软件需求的整体理解,并提供对信息域、功能和系统行为的深入理解。
- 关于基础结构和其他非功能的模型应推延到设计阶段再考虑。例如,可能需要一个数据库,但是只有在已经完成问题域分析之后才应考虑实现数据库所必需的类、访问数据库所需的功能以及使用时所表现出的行为。
- 最小化整个系统内的关联。表现类和功能之间的联系非常重要,但是,如果“互联”的层次非常高,则应该想办法减少互联。
- 确认需求模型为所有利益相关者都带来价值。对模型来说,每个客户都有自己的使用目的。例如,业务人员将使用模型确认需求,设计人员将使用模型作为设计的基础,质量保证人员将使用模型帮助规划验收测试。
- 尽可能保持模型简洁。如果没有提供新的信息,就不要添加附加图表;如果一个简单列表够用,就不要使用复杂的表示方法。

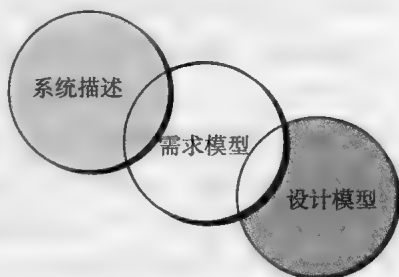


图9-1 需求模型在系统描述和设计模型之间建立桥梁

关键点 分析模型应该描述什么是客户所需。应该建立设计的基础,建立有效的目标。

提问 进行需求分析时有没有可以帮助我们指南?

引述 值得进攻的问题总是通过反击证明其价值。
Piet Hein

168

169

9.1.3 域分析

在需求工程讨论中(第8章),我们注意到分析模式通常在特定业务领域内的很多应用系统中重复发生。如果用一种方式对这些模式加以定义和分类,让软件工程师或分析师识别并复用这些模式,将促进分析模型的创建。更重要的是,应用可复用的设计模式和可执行的软件构件的可能性将显著增加。这将把产品投放市场的时间提前,并减少开发费用。

但问题是,首先如何识别分析模式?由谁来对分析模式进行定义和分类,并为随后的项目准备好分析模式?这些问题的答案在域分析中。Firesmith [Fir93] 这样描述域分析:

软件域分析是指识别、分析和详细说明某个特定应用领域的共同需求,特别是那些在该应用领域内被多个项目重复使用的……(面向对象的域分析是)在某个特定应用领域内,根

网络资源 很多关于域分析的有用信息可以在 www.sei.cmu.edu 中找到。

169

据通用的对象、类、部件和框架，识别、分析和详细说明公共的、可复用的能力。

“特定应用领域”的范围从航空电子设备到银行业，从多媒体视频游戏到医疗设备中的嵌入式软件。域分析的目标很简单：查找或创建那些广泛应用的分析类或分析模式，使其能够复用。^①

使用本书前面介绍的术语，域分析可以被看作软件过程的一个普适性活动。意思是域分析是正在进行的软件工程活动，而不是与任何一个软件项目相关的。域分析师的角色有些类似于重型机械制造业中一名优秀的刀具工的角色。刀具工的工作是设计并制造工具，这些工具可被很多人用来进行类似的而不一定是同样的工作。域分析师^②的角色是发现和定义可复用的分析模式、分析类和相关信息，这些也可用于类似但不要求必须是完全相同的应用。

关键点 域分析不关注特定的应用系统，而是关注应用所属的领域。其目的在于识别那些解决可用于域内所有应用系统的共同问题。

图 9-2[Arn89] 说明了域分析过程的关键输入和输出。应该调查领域知识的来源以便确定可以在整个领域内复用的对象。

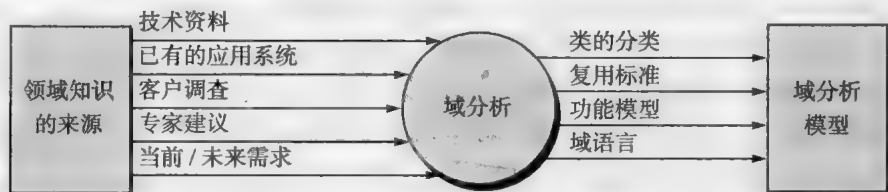


图 9-2 域分析的输入和输出

SafeHome 域分析

[场景] Doug Miller 的办公室，在销售业务会议之后。

[人物] Doug Miller，软件工程经理；Vinod Raman，软件工程团队成员。

[对话]

Doug：我需要你做一个特殊的项目，Vinod。我将会把你调离需求收集会议。

Vinod（皱眉不悦）：太糟了。这可行吗……我已从中获取了一些需求。出什么事啦？

Doug：Jamie 和 Ed 将接管你的工作。不管怎样，市场部坚持要我们在第一次发布的 SafeHome 中交付具有互联网能力的家庭安全功能。我们一直紧张地为此工作着……

没有足够的时间和人力，所以我们马上要解决 PC 接口和 Web 接口两个问题。

Vinod（看上去很疑惑）：我不知道原先设定的计划……我们甚至还没有完成需求收集。

Doug（无精打采地微笑）：我知道，但时间太紧了，我决定马上和市场部开始战略合作……无论如何，一旦从所有需求收集会议上获得信息，我们就将重新审视任何不确定的计划。

Vinod：好的，会发生什么事？你要我做什么事吗？

Doug：你知道“域分析”吗？

Vinod：略知一些。在建立应用系统时为

① 域分析的一个补充观点是：“包括为域建模，因此软件工程师和其他的利益相关者可以更好地学习……不是所有域的都必然导致可复用的类。” [Let03]。

② 不要认为有域分析员在工作，软件工程师就不需要理解应用问题的领域。软件团队的每个成员都应该一定程度地了解软件将要工作的领域。

做同一件事的应用系统寻找相似的模式。如果可能，可以在工作中剽窃这些模式并复用它们。

Doug：我觉得用剽窃这个词不太恰当，但你的意思基本是正确的。我想让你做的事是开始研究可控制 SafeHome 这类系统的现存用户接口。我想你需要组织一套模式和分析类，它们通常既能坐在房间里处理基于 PC 的接口，也能处理通过互联网进入的基于浏览器的接口。

Vinod：把它们做成相同的东西可以节省时间……为什么不这么做呢？

Doug：哦……。有你这种想法的人真是非

常好。整个核心要点是如果能识别出两种接口，那么就采用相同的代码，等等，这样我们就节省了时间和人力。这些正是市场部坚持的。

Vinod：那你想要什么？类，分析模式，设计模式？

Doug：所有。非正式地说这就是关键所在。我就是想稍早一些开始我们的内部分析和设计工作。

Vinod：我将在类库中看看我们已经得到了哪些。我也会使用几个月前我读的一本书中的模式模版。

Doug：太好了，继续工作吧。

9.1.4 需求建模的方法

一种考虑数据和处理的需求建模方法称作结构化分析，其中处理过程将数据作为独立实体加以转换。数据对象建模定义了对对象的属性和关系，操作数据对象的处理建模应表明当数据对象在系统内流动时处理过程将如何转换数据。

需求建模的第二种方法称作面向对象的分析，这种方法关注类的定义和影响客户需求的类之间的协作方式。UML 和统一过程（第 4 章）主要是面向对象的分析方法。

在本书这一版中，我们已经选择强调采用面向对象分析的元素进行 UML 建模。目的是建议一种联合的表达方式，给项目利益相关者提供最好的软件需求，以便为软件设计提供桥梁。

如图 9-3 所示，需求模型的每个元素表示源自不同观点的问题。基于场景的元素表述用户如何与系统和使用软件时出现的特定活动序列进行交互。基于类的元素的内容包括：系统操作的对象，应用在这些对象间影响操作和对对象间关系（某层级）的操作，以及定义

的类间发生的协作。行为元素描述了外部事件如何改变系统或驻留在系统里的类的状态。最后，面向流的元素表示信息转换的系统，描述了数据对象在流过各种系统功能时是如何转换的。

需求模型导出每个建模元素的派生类。然而，每个元素（即用于构建元素和模型的图表）的特定内容可能因项目而异。就像我们在本书中多次提到的那样，软件团队必须想办法保持模型的简单性。只有那些为模型增加价值的建模元素才能使用。

引述 分析容易使人灰心丧气，全都是非常复杂的人际关系，不确定且困难的东西。总而言之，分析让人着迷。一旦沉迷，原来轻松构建系统的快乐将难以令你满足。

Tom DeMarco

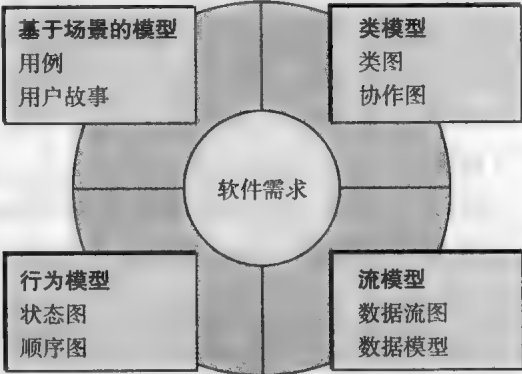


图 9-3 需求模型的元素

9.2 基于场景建模

尽管可以用多种方式度量基于计算机的系统或产品的成果，但用户的满意度仍是其中最重要的。如果软件工程师了解最终用户（和其他参与者）希望如何与系统交互，软件团队将能够更好、更准确地刻画需求特征，完成更有针对性的分析和设计模型。因此，使用 UML^①需求建模将从开发用例、活动图和泳道图形式的场景开始。

提问 在描述需求模型时常用哪些不同的观点？

9.2.1 创建初始用例

Alistair Cockburn 刻画了一个名为“合同行为”的用例 [Coc01b]。我们在第 8 章讨论的“合同”定义了一个参与者^②使用基于计算机的系统完成某个目标的方法。本质上用例捕获了信息的产生者、使用者和系统本身之间发生的交互。在本节，我们研究如何开发用例，这是分析建模活动的一部分^③。

引述 (用例)只是帮助定义系统之外(参与者)存在什么以及系统应完成什么(用例)。

Ivar Jacobson

第 8 章中我们已经提到，用例从某个特定参与者的角度出发，采用简明的语言描述一个特定的使用场景。但是我们如何知道：(1) 编写什么？(2) 写多少？(3) 编写说明应该多详细？(4) 如何组织说明？如果想让用例像一个需求建模工具那样提供价值，那么必须回答这些问题。

关键点 在某些情况下，用例成为最主要的需求工程机制，但是这并不意味着你应该放弃适用的其他建模方法。

编写什么？两个首要的需求工程工作——起始和获取——提供了开始编写用例所需要的信息。运用需求收集会议、质量功能部署 (Quality Function Deployment, QFD) 和其他需求工程机制确定利益相关者，定义问题的范围，说明整体的运行目标，建立优先级顺序，概述所有已知的功能需求，描述系统将处理的信息 (对象)。

开始开发用例时，应列出特定参与者执行的功能或活动。这些可以借助所需系统功能的列表，通过与利益相关者交流，或通过评估活动图 (作为需求建模中的一部分而开发) 获得 (9.3.1 节)。

173

SafeHome 开发另一个初始用户场景

[场景] 会议室，第二次需求收集会议中。

能开发一个用户场景。

[人物] Jamie Lazar 和 Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

Jamie: 谁在其中扮演参与者的角色？

主持人: 我想 Meredith (市场营销人员) 已经在该功能上进行了一些工作，你来试试这个角色吧。

[对话]

Meredith: 你想采用我们上次用的方法，是吗？

主持人: 现在是我们开始讨论 SafeHome 监视功能的时候了，让我们为访问监视功

主持人: 是的，同样的方法。

① UML 是本书通篇使用的建模符号。附录 1 为那些不熟悉 UML 基本符号的读者提供了简要指南。

② 参与者不是一个确定的人员，而是人员 (或设备) 在特定的环境内所扮演的一个角色，参与者“呼叫系统并由系统提供一种服务” [Coc01b]。

③ 用例是用户接口的分析建模中特别重要的一部分，我们将在第 15 章详细讨论接口分析。

Meredith：好的，很明显，开发监视功能的理由是允许房主远距离检查房屋、记录并回放捕获的录像……就是这样。

Ed：我们采用压缩的方法存储图像吗？

主持人：好问题，但是我们现在先不考虑实现的问题，Meredith，你说呢？

Meredith：好的，这样对于监视功能基本上就有两部分……第一部分是配置系统，包括布置建筑平面图——我们需要工具来帮助房主做这件事，第二部分是实际的监视功能本身。因为布局是配置活动的一部分，所以我将重点集中在监视功能。

主持人（微笑）：抢先说出我想说的话。

Meredith：哦……我希望通过电脑或通过 Internet 访问监视功能。我的感觉是 Internet 访问可能使用的频率更高一些。

不管怎样，我希望能够在计算机上和控制面板上显示摄像机图像并移动某个摄像机镜头。在房屋平面设计图上可以选择指定摄像机，我希望可以有选择地记录摄像机输出和回放摄像机输出，我还希望能够使用特殊的密码阻止对某个或多个摄像机的访问。希望有支持小窗口显示形式的选项，即从所有的摄像机显示图像，并能够选择某一个进行放大。

Jamie：那些叫作缩略视图。

Meredith：对，然后我希望从所有摄像机获得缩略视图。我也希望监视功能的接口和所有其他的 SafeHome 接口有相同的外观和感觉。

主持人：干得好，现在，让我们更详细地讨论这个功能……

上面讨论的 SafeHome 住宅监视功能（子系统）确定了如下由参与者房主执行的功能（简化列表）：

- 选择将要查看的摄像机。
- 提供所有摄像机的缩略视图。
- 在计算机的窗口中显示摄像机视图。
- 控制某个特定摄像机的镜头转动和缩放。
- 可选择地记录摄像机的输出。
- 回放摄像机的输出。
- 通过 Internet 访问摄像机监视功能。

174

随着和利益相关者（扮演房主的人）交谈的增多，需求收集团队将为每个标记的功能开发用例。通常，用例首先用非正式的描述性风格编写。如果需要更正式一些，可以使用类似于第8章中提出的某个结构化的形式重新编写同样的用例，在本节的后面我们将进行重新生成。

为了举例说明，考虑“通过互联网访问摄像机监视设备—显示摄像机视图（Access Camera Surveillance-Display Camera Views, ACS-DCV）”功能，扮演参与者房主的利益相关者可能会编写如下说明。

用例：通过互联网访问摄像机监视设备—显示摄像机视图（ACS-DCV）。

参与者：房主。

如果我正在进行远程访问，那么我可以使用任何计算机上的合适的浏览器软件登录 SafeHome 产品网站。输入我的账号和两级密码，一旦被确认，我可以访问已安装的 SafeHome 系统的所有功能。为取得某个摄像机视图，从显示的主功能按钮中选择“监视”，然后选择“选取摄像机”，这时将会显示房屋的平面设计图，之后再选择感兴趣的摄像机。另一种可选方法是，通过选择“所有摄像机”同时从所有的摄像机查看缩略视图快照。当选

择了某个摄像机时，可以选择“查看”，然后以每秒一帧速度显示的图像就可以在窗口中显示。如果希望切换摄像机，则选择“选取摄像机”，这时原来窗口显示的信息消失，并且再次显示房间的平面设计图，然后就可以选择感兴趣的摄像机，以便显示新的查看窗口。

描述性用例的一种表达形式是通过用户活动的顺序序列表现交互，每个行动由声明性的语句表示。再以 ACS-DCV 功能为例，我们可以写成如下形式。

用例：通过互联网访问摄像机监视设备 - 显示摄像机视图 (ACS-DCV)。

参与者：房主。

1. 房主登录 SafeHome 产品网站。
2. 房主输入账号。
3. 房主输入两个密码（每个至少 8 个字符长度）。
4. 系统显示所有的主要功能按钮。
5. 房主从主要功能按钮中选择“监视”。
6. 房主选择“选取摄像机”。
7. 系统显示房屋的平面设计图。
8. 房主从平面设计图中选择某个摄像机图标。
9. 房主选择“视图”按钮。
10. 系统显示一个由摄像机编号确定的视图窗口。
11. 系统在视图窗口内以每秒一帧的速度显示视频输出。

引述 用例可以应用在许多（软件）过程中，我们感兴趣的是迭代和风险驱动过程。

Geri Schneider,
Jason Winters

175

注意，这个连续步骤的陈述没有考虑其他可能的交互（描述更加自由随意而且确实表达了一些其他选择）。这种类型的用例有时被称作主场景 [Sch98a]。

9.2.2 细化初始用例

为了全面理解用例描述功能，对交互操作给出另外的描述是非常有必要的。

因此，主场景中的每个步骤将通过如下提问得到评估 [Sch98a]：

- 在这一步，参与者能做一些其他动作吗？
- 在这一步，参与者有没有可能遇到一些错误条件？如果有可能，这些错误会是什么？
- 在这一步，参与者有没有可能遇到一些其他行为（如由一些参与者控制之外的事件调用）？如果有，这些行为是什么？

提问 在开发场景用例时，如何检查动作的可选过程？

这些问题的答案导致创建一组次场景，次场景属于原始用例的一部分，但是表现了可供选择的行为。例如，考虑前面描述的主场景的第 6 和第 7 步。

6. 房主选择“选取摄像机”。
7. 系统显示房屋的平面设计图。

在这一步，参与者能做一些其他动作吗？答案是肯定的。考虑自由随意的操作方式，参与者可以选择同时查看所有摄像机的缩略视图。因此，一个次场景可能是“查看所有摄像机的缩略视图”。

在这一步，参与者有没有可能遇到一些错误条件？作为基于计算机的系统操作，任何数量的错误条件都可能发生。在该语境内，我们仅仅考虑在第 6 和第 7 步中说明的活动的直接错误条件，问题的答案还是肯定的。带有摄像机图标的房屋平面图可能还没有配置过，这样

选择“选取摄像机”就导致错误的条件：“没有为该房屋配置平面设计图”^①。该错误条件就成为一个次场景。

在这一步，参与者有没有可能遇到一些其他行为？问题的答案再一次是肯定的。当第6和第7步发生时，系统可能遇到报警。这将导致系统显示一个特殊的报警通知（类型、地点、系统动作），并向参与者提供和报警性质相关的一组操作。因为这个次场景可以在所有的实际交互中发生，所以不会成为 ACS-DCV 用例的一部分。而且，我们将开发一个单独的用例——遇到报警条件——这个用例可以被其他用例引用。

前面段落描述的每种情景都是以客户用例的异常处理为特征的。异常处理描述了这样一种情景（可能是失败条件或参与者选择了替代方案），该场景导致系统展示出某些不同的行为。

提问 什么是用例异常？我们如何决定这些异常的样子？

Cockburn[Coc01b] 推荐使用“头脑风暴”来推动团队合理地完成每个用例中一系列的异常处理。除了本节前面提到了三个常规问题外，还应该研究下面的问题：

- 在这个用例中是否有某些具有“确认功能”的用例出现？包括引用确认功能，以及可能出现的出错条件。
- 在这些用例中是否有支持功能（或参与者）的应答失败？例如，某个用户动作是等待应答，但该功能已经应答超时了。
- 性能差的系统是否会导致无法预期或不正确的用户活动？例如，一个基于 Web 的接口应答太慢，导致用户在处理按钮上已经做了多重选择。这些选择队列最终不恰当地生成了一个出错条件。

其他问题和回答将继续扩充这份列表，使用下面的标准可使这些问题合理化 [Co01b]：用例应该注明异常处理，即如果软件能检测出异常所发生的条件就应该马上处理这个条件。在某些情况下，异常处理可能拖累其他用例处理条件的开发。

9.2.3 编写正式用例

9.2.1 节表述的非正式用例对于需求建模常常是够用的。但是，当用例需要包括关键活动或描述一套具有大量异常处理的复杂步骤时，我们就会希望采用更为正式的方法。

在 SafeHome 中的 ACS-DCV 用例把握了正式用例的典型描述要点。在以下的 SafeHome 中：情境目标确定了用例的全部范围。前提条件描述在用例初始化前应该知道哪些信息。触发器确定“用例开始”的事件或条件 [Coc01b]。场景列出参与者和恰当的系统应答所需要的特定活动。异常处理用于细化初始用例时没有涉及的情景（9.2.2 节）。此外还可能包含其他主题，并给出合理的自我解释。

177

SafeHome 监视的用例模板

用例：通过互联网访问摄像机监视设备—显示摄像机视图（ACS-DCV）。

迭代：2。最新更改记录：V. Raman，1 月 14 日。

主参与者：房主。

情境目标：从任何远程地点通过互联网查看遍布房间的摄像头输出。

前提条件：必须完整配置系统；必须获得

^① 在该例子中，另一个参与者——系统管理员必须配置平面设计图、安装并初始化（如分配设备编号）所有的摄像头，并且通过系统平面设计图访问和测试每个摄像头能达到的特定作用。

正确的账号和密码。

触发器：房主在远离家的时候决定查看房屋内部。

场景：

1. 房主登录 SafeHome 产品网站。
2. 房主输入他的账号。
3. 房主输入两个密码（每个都至少有 8 个字符的长度）。
4. 系统显示所有的主要功能按钮。
5. 房主从主要功能按钮中选择“监视”。
6. 房主选择“选取摄像机”。
7. 系统显示房屋的平面设计图。
8. 房主从房屋的平面设计图中选择某个摄像机的图标。
9. 房主选择“视图”按钮。
10. 系统显示一个由摄像机编号确定的视图窗口。
11. 系统在视图窗口中以每秒一帧的速度显示视频输出。

异常处理：

1. 账号或密码不正确或不被确认——参看用例“确认账号和密码”。
2. 没有为该系统配置监视功能——系统显示恰当的错误消息，参看用例“配置监视功能”。
3. 房主选择“查看所有摄像机的缩略视图

快照”——参看用例“查看所有摄像机的缩略视图快照”。

4. 平面设计图不可用或是还没有配置——显示恰当的错误消息，参看用例“配置平面设计图”。
5. 遇到报警条件——参看用例“遇到报警条件”。

优先级：必须在基础功能之后实现中等优先级。

何时有效：第三个增量。

使用频率：频率较低。

参与者的连接渠道：通过基于个人计算机的浏览器和互联网连接到 SafeHome 网站。

次要参与者：系统管理员，摄像机。

次要参与者的连接渠道：

1. 系统管理员：基于个人计算机的系统。
2. 摄像机：无线连接。

未解决的问题：

1. 用什么机制保护 SafeHome 产品的雇员在未授权的情况下能使用该功能？
2. 足够安全吗？黑客入侵该功能将使最主要的个人隐私受侵。
3. 在给定的摄像机视图所要求的带宽下，可以接受通过互联网的系统响应吗？
4. 若可以使用高带宽的连接，能开发出比每秒一帧更快的视频速度吗？

[178]

在很多情况下，不需要创建图形化表示的用户场景。然而，在场景比较复杂时，图表化的表示更有助于理解。正如我们在本书前面所提到的，UML 的确提供了图形化表现用例的能力。图 9-4 为 SafeHome 产品描述了一个初步的用例图，每个用例由一个椭圆表示。本节仅详细讨论了 ACS-DCV。

每种建模注释方法都有其局限性，用例方法也不例外。和其他描述形式一样，用例的好坏取决于它的描述者。如果描述不清晰，用例可能会误导或有歧义。用例关注功能和行为需求，一般不适用于非功能需求。对于必须特别详细和精准的需求建模情境（例如安全关键系统），用例方法就不够用了。

然而，软件工程师遇到的绝大多数情境都适用基于场景建模。如果开发得当，用例作为一个建模工具将带来很多益处。

网络资源 什么时候结束用例编写？关于该话题有价值的论述可以参阅 ootips.org/use-cases-done.html。

9.3 补充用例的 UML 模型

很多基于文本的需求建模情景（即使和用例一样简单）不能简明扼要地传递信息。在这种情况下，你应能从大量的 UML 图形模型中进行选择。

9.3.1 开发活动图

UML 活动图在特定场景内通过提供迭代流的图形化表示来补充用例。类似于流程图，活动图使用两端为半圆形的矩形表示一个特定的系统功能，箭头表示通过系统的流，菱形表示分支（标记从菱形发出的每个箭头），实水平线意味着并行发生的活动。ACS-DCV 用例的活动图如图 9-5 所示。应注意到活动图增加了额外的细节，而这些细节是用例不能直接描述的（隐含的）。例如，用户可以尝试有限次数地输入账号和密码，这可以通过“提示重新输入”的判定菱形来体现。



图 9-4 SafeHome 系统的初步用例图

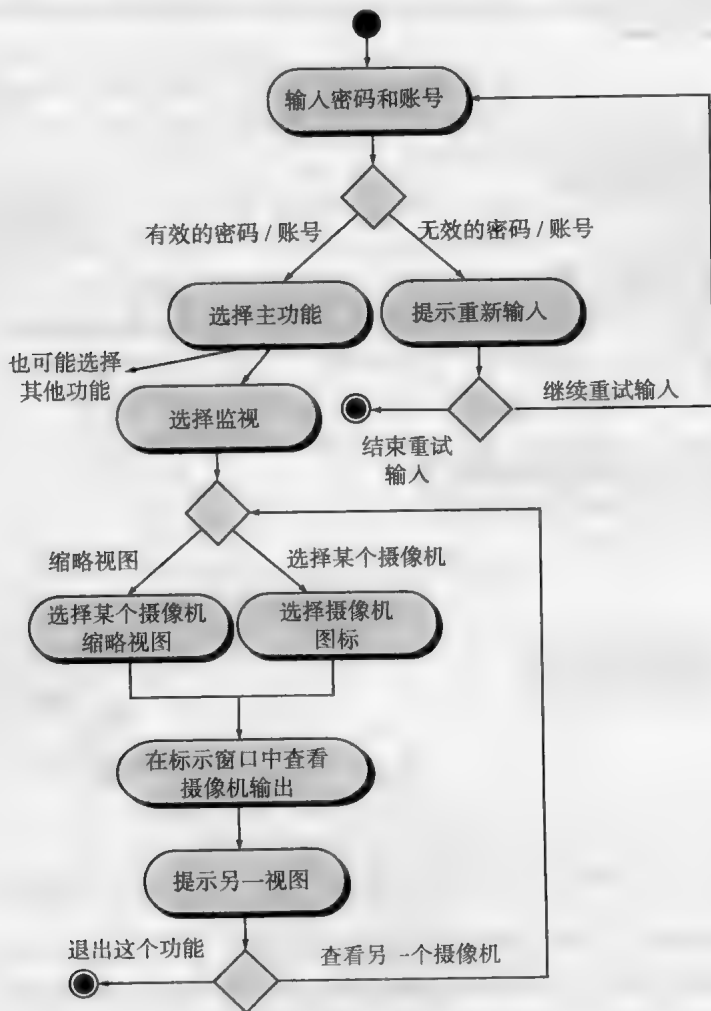


图 9-5 通过互联网访问摄像机监视设备并显示摄像机视图功能的活动图

9.3.2 泳道图

UML 泳道图是活动图的一种有用的变形，允许建模人员表示用例所描述的活动流，同时指出哪个参与者（如果在某个特定用例中涉及了多个参与者）或分析类（第 10 章）负责由活动矩形所描述的活动。职责由纵向分割图中的并行条表示，就像游泳池中的泳道。

关键点 UML 泳道图表现了活动流和一些判定，并指明由哪个参与者实施。

181

三种分析类——房主、摄像机和接口——对于图 9-5 所表示的活动图中的情景具有直接或间接的责任。参看图 9-6，重新排列活动图，和某个特殊分析类相关的活动按类落入相应的泳道中。例如，接口类表示房主可见的用户接口。活动图标记出对接口负责的两个提示——“提示重新输入”和“提示另一视图”。这些提示以及与此相关的判定都落入了接口泳道。但是，从该泳道发出的箭头返回到房主泳道，这是因为房主的活动在房主泳道中发生。

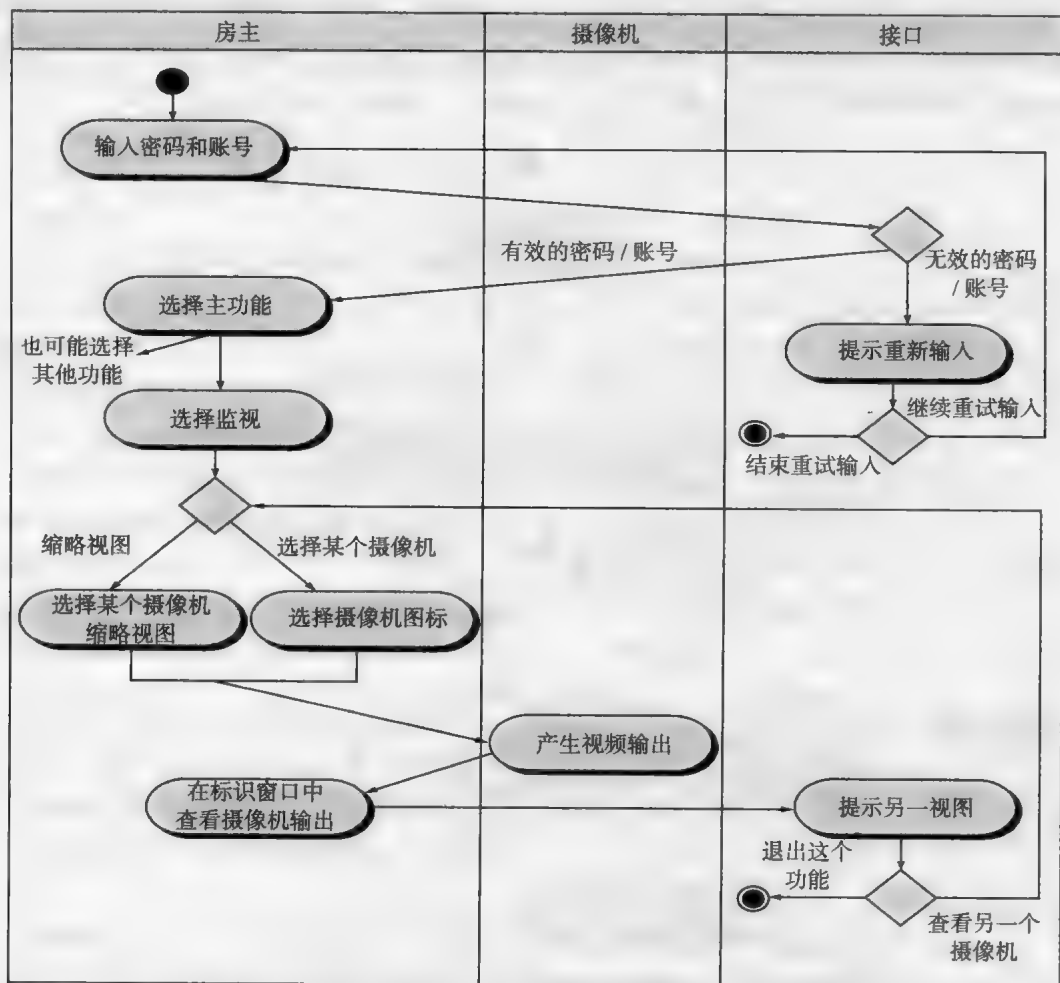


图 9-6 通过互联网访问摄像机监视设备并显示摄像机视图功能的泳道图

借助活动图和泳道图，面向过程的用例表示出各种参与者行使的一些特定功能（或其他处理步骤），以便满足系统需求。但是需求的过程视图仅表示系统的单一维度，在第 10 章和第 11 章，我们将考察需求建模的其他维度。

引述 好模型引导你思考，而坏模型会扭曲它。

Brian Marick

9.4 小结

需求建模的目标是创建各种表现形式，用其描述什么是客户需求，为生成软件设计建立基础，一旦软件建立，这些需求将用于验证。需求模型在系统级表示层和软件设计之间构造了桥梁。系统表示层描述了整个系统和业务功能，软件设计描述了软件应用的架构、用户接口和组件级的结构。

基于场景的模型从用户的角度描述软件需求。用例是主要的建模元素，它以叙述方式或以模板驱动方式描述了参与者和软件之间的交互活动。在需求获取过程中得到的用例定义了特定功能或交互活动的关键步骤。用例的形式化和详细程度各不相同，但其最终结果为所有的其他分析建模活动提供了必需的输入。还可以使用活动图说明场景，即一种类似于流程图的图形表现形式，描述在特定场景中的处理流。泳道图显示了如何给不同的参与者或类分配处理流。

习题与思考题

- 9.1 有没有可能在分析模型创建后立即开始编码？解释你的答案，然后说服反方。
- 9.2 一个单凭经验的分析原则指出模型“应该关注在问题域或业务域中可见的需求”。在这些域中哪些类型的需求是不可见的？提供一些例子。
- 9.3 域分析的目的是什么？如何将域分析与需求模式概念相联系？
- 9.4 有没有可能不完成图 9-3 所示的四种元素就开发出一个有效的分析模型？解释一下。
- 9.5 某个大城市的公共工程部决定开发基于 Web 的路面坑洼跟踪和修补系统（PHTRS）。说明如下：

182

市民可以登录 Web 站点报告路面坑洼的地点和严重程度。上报后，该信息将记入“路面坑洼跟踪和修补系统”，分配一个标识号，保存如下信息：街道地址、大小（比例从 1 到 10）、位置（中央、路边等）、地区（由街道地址确定）以及修补优先级（由坑洼大小确定）。工作订单数据和每个坑洼有关联，数据包含坑洼位置和大小、维修组标识号、维修组内人员数量、分配的设备、修复耗时、坑洼状态（正在处理中、已修复、临时修复、未修复）、使用的填充材料数量以及修复成本（根据修复耗时、人员数量、材料和使用的设备计算）。最后，生成损失文件以便保存该坑洼所造成的损失报告信息，并包含公民的姓名、地址、电话号码、损失类型、损失金额。PHTRS 基于在线系统，可交互地进行所有查询。

为 PHTRS 系统画出 UML 用例图，你必须对用户和系统的交互方式做一些假设。

- 9.6 编写 2～3 个用例描述 PHTRS 系统中的各种参与者的角色。
- 9.7 为 PHTRS 系统的某个部分开发一个活动图。
- 9.8 为 PHTRS 系统的一个或多个部分开发一个泳道图。

扩展阅读与信息资源

用例是为所有需求建模方法服务的基础。讨论这一主题的书很多，包括：Gomaa（《Software Modeling：UML，Use Case，Patterns，and Architecture》，Cambridge University Press，2011），Rosenberg 和 Stephens（《Use Case Driven Object Modeling with UML：Theory and Practice》，Apress，2007），Denny（《Succeeding with Use Cases：Working Smart to Deliver Quality》，Addison-Wesley，2005），Alexander 和 Maiden（eds.）（《Scenarios，Stories，Use Cases：Through the Systems Development Life-Cycle》，Wiley，2004），Bittner 和 Spence（《Use Case Modeling》，Addison-Wesley，2002），Cockburn[Coc01b]。其他参考资料请关注第 8 章。

UML 建模技术可以应用于分析和设计，讨论这方面的书包括：Dennis 和他的同事（《Systems

Analysis and Design with UML Version 2.0》, 4th ed., Wiley, 2012), O'Docherty (《Object-Oriented Analysis and Design: Understanding System Development with UML2.0》, Wiley, 2005), Arlow 和 Neustadt (《UML 2 and the Unified Process》, 2nd ed., Addison-Wesley, 2005), Roques(《UML in Practice》, Wiley, 2004), Larman(《Applying UML and Patterns》, 2nd ed., Prentice-Hall, 2001), Rosenberg 和 Scott (《Use Case Driven Object Modeling with UML》, Addison-Wesley, 1999)。

关于需求的书包括 Robertson(《Mastering the Requirements Process: Getting Requirements Right》, 3rd ed., Addison-Wesley, 2012), Hull、Jackson 和 Dick (《Requirements Engineering》, 3rd ed., Springer, 2010), Alexander 和 Beus Dukic(《Discovering Requirements: How to Specify Products and Services》, Wiley, 2009)。

网上有很多关于需求建模的信息。可以参考 SEPA 网站 www.mhhe.com/pressman 上有关分析建模的最新参考文献。

需求建模：基于类的方法

要点浏览

概念：软件问题总是以一套交互对象为特征，借此在系统内表达每一个感兴趣的事物。每个对象变成对象类中的一个成员。对象的状态描述了每个对象，即数据属性描述了对象。我们可以用基于类的需求建模方法表达上述所有内容。

人员：软件工程师（有时被称作分析师）使用从客户那里获取的需求来建立基于类的模型。

重要性：基于类的需求模型利用客户视角下应用或系统中的对象。这个模型描述了常规客户的系统视图。因此，客户能恰当地进行评估，并尽早获得有用的反馈信息。然后，在重新定义模型时，它将成为软件设计的基础。

步骤：基于类的建模定义了对对象、属性和关系。对一个问题描述做一番简单的探究后，能从问题的陈述中开发外部对象和类，并以基于文本或图表的形式进行表达。在创建了模型的雏形以后，就要对其不断改进，分析和评估其清晰性、完整性和一致性。

工作产品：可以为需求建模选择大量的基于文本和图表形式的分析模型，每种表达方法都提供了一个或多个模型元素的视图。

质量保证措施：必须评审需求建模工作产品的正确性、完整性和一致性。必须反映所有利益相关者的要求并建立一个可以从中导出设计的基础。

20 世纪 90 年代早期，第一次引入基于类的需求建模方法时，常以面向对象分析作为分类方法。虽然有大量不同的基于类的方法和表达方式，但 Coad 和 Yourdon[Coa91] 为所有人注明了其统一特征：

面向对象方法基于的都是我们最初在幼儿园中学到的内容：对象和属性，全局和部分，类和成员。

在需求建模中使用基于类的方法可以精巧地表达这些常规内容，使非技术型的利益相关者理解项目。随着需求模型的细化和扩展，它还将包含一份规格说明，以帮助软件工程师创建软件的设计部分。

基于类建模表示了系统操作的对象、应用于对象间能有效控制的操作（也称为方法或服务）、这些对象间（某种层级）的关系以及已定义类之间的协作。基于类的分析模型的元素包括类和对象、属性、操作、CRC 模型、协作图和包。下面几节将提供一系列有助于识别和表示这些元素的非正式指导原则。

关键概念

分析类

分析包

关联

属性

协作性

CRC 建模

依赖性

语法解析

运维

职责

10.1 识别分析类

当你环顾房间时，就可以发现一组容易识别、分类和定义（就属性和操作而言）的物理

对象。但当你“环顾”软件应用的问题空间时，了解类（和对象）就没有那么容易了。

通过检查需求模型（第9章）开发的使用场景，并对系统开发的用例进行“语法解析”[Abb83]，我们就可以开始进行类的识别了。带有下列线的每个名词或名词词组可以确定为类，将这些名词输入到一个简单的表中，并标注出同义词。如果要求某个类（名词）实现一个解决方案，那么这个类就是解决方案空间的一部分；否则，如果只要求某个类描述一个解决方案，那么这个类就是问题空间的一部分。

一旦分离出所有的名词，我们该寻找什么？分析类表现为如下方式之一。

- 外部实体（例如其他系统、设备、人员）：产生或使用基于计算机系统的信息。
- 事物（例如报告、显示、字母、信号）：问题信息域的一部分。
- 偶发事件或事件（例如所有权转移或完成机器人的一组移动动作）：在系统操作环境中发生。
- 角色（例如经理、工程师、销售人员）：由和系统交互的人员扮演。
- 组织单元（例如部门、组、团队）：和某个应用系统相关。
- 场地（例如制造车间或码头）：建立问题的环境和系统的整体功能。
- 结构（例如传感器、四轮交通工具、计算机）：定义了对象的类或与对象相关的类。

这种分类只是文献中已提出的大量分类之一^①。例如，Budd[Bud96]建议了一种类的分类

法，包括数据产生者（源点）、数据使用者（汇点）、数据管理者、查看或观察者类以及帮助类。

还需要特别注意的是：什么不能是类或对象。通常，决不应该使用“命令过程式的名称”为类命名[Cas89]。例如，如果医疗图像系统的软件开发人员使用名字“InvertImage”甚至“ImageInversion”定义对象，就可能犯下一个小小的错误。从软件获得的Image当然可能是一个类（这是信息域中的一部分），图像的翻转是适用于该对象的一个操作，很可能将翻转定义为对于对象Image的一个操作，但是不可能定义单独的类来暗示“图像翻转”。如Cashman[Cas89]所言：“面向对象的目的是封装，但仍保持独立的数据以及对数据的操作。”

为了说明在建模的早期阶段如何定义分析类，考虑对SafeHome安全功能的“处理说明”^②进行语法解析（对第一次出现的名词加下划线，第一次出现的动词采用斜体）。

SafeHome安全功能 允许房主 在安装时配置安全系统，监控所有连接到安全系统的传感器，通过互联网、计算机或控制面板和房主交互信息。

在安装过程中，用SafeHome个人计算机来设计和配置系统。为每个传感器分配编号和类型，用主密码控制启动和关闭系统，而且当传感器事件发生时会拨打输入的电话号码。

识别出一个传感器事件时，软件激活装在系统上的发声警报，由房主在系统配置活动中指定的延迟时间结束后，软件拨打监控服务的电话号码并提供位置信息，报告检测到的事件性质。电话号码将每隔20秒重拨一次，直至电话接通。

引述 真正困难的问题是首先发现什么才是正确的对象（类）。

Carl Argila

提问 分析类如何把自己表现为解决方案空间的元素？

建议 虽然语法解析不能保证万无一失，但如果你正在定义数据对象及其操作的转变，语法解析会让你飞跃上一个非常出色的起始点。

① 另一个重要的分类是指定义实体、边界和控制类，在10.5节中讨论。

② “处理说明”类似于用例的风格，但目标稍有不同。“处理说明”提供了将要开发的功能的整体说明，而不是从某个参与者的角度写的场景。但是要注意很重要的一点，在需求收集（获取）部分也会使用语法解析来开发每个用例。

房主通过控制面板、个人计算机或浏览器（统称为接口）来接收安全信息。接口在控制面板、计算机或浏览器窗口中显示提示信息和系统状态信息。房主采用如下形式进行交互活动……

186

抽取这些名词，可以获得如下表所示的一些潜在类：

潜在类	一般分类	潜在类	一般分类
房主	角色或外部实体	主密码	事物
传感器	外部实体	电话号码	事物
控制面板	外部实体	传感器事件	事件
安装	事件	发声警报	外部实体
系统（别名安全系统）	事物	监控服务	组织单元或外部实体
编号，类型	不是对象，是传感器的属性		

这个表应不断完善，直到已经考虑到了处理说明中所有的名词。注意，我们称列表中的每一输入项为“潜在”对象，在进行最终决定之前还必须对每一项都深思熟虑。

Coad 和 Yourdon[Coa91] 建议了 6 个选择特征，在分析模型中，分析师考虑每个潜在类是否应该使用如下这些特征。

提问 如何确定某个潜在类是否应该真的成为一个分析类？

1. 保留信息。只有记录潜在类的信息才能保证系统正常工作，这样潜在类才能在分析过程中发挥作用。
2. 所需服务。潜在类必须具有一组可确认的操作，这组操作能用某种方式改变类的属性值。
3. 多个属性。在需求分析过程中，焦点应在于“主”信息；事实上，只有一个属性的类可能在设计中有用，但是在分析活动阶段，最好把它作为另一个类的某个属性。
4. 公共属性。可以为潜在类定义一组属性，这些属性适用于类的所有实例。
5. 公共操作。可以为潜在类定义一组操作，这些操作适用于类的所有实例。
6. 必要需求。在问题空间中出现的外部实体，以及任何系统解决方案运行时所必需的生产或消费信息，几乎都被定义为需求模型中的类。

考虑包含在需求模型中的合法类，潜在类应全部（或几乎全部）满足这些特征。判定潜在类是否应包含在分析模型中多少有点主观，而且后面的评估可能会舍弃或恢复某个类。然而，基于类建模的首要步骤就是定义类，因此必须进行决策（即使是主观的）。以此为指导，根据上述选择特征进行了筛选，分析师列出 SafeHome 潜在类，如下表所示。

引述 阶级斗争中，一些阶级胜利了，一些阶级被消灭了……
毛泽东

187

潜在类	适用的特征编号
房主	拒绝：6 适用，但是 1、2 不符合
传感器	接受：所有都适用
控制面板	接受：所有都适用
安装	拒绝
系统（别名安全系统）	接受：所有都适用
编号，类型	拒绝：3 不符合，这是传感器的属性
主密码	拒绝：3 不符合
电话号码	拒绝：3 不符合
传感器事件	接受：所有都适用
发声警报	接受：2、3、4、5、6 适用
监控服务	拒绝：6 适用，但是 1、2 不符合

应注意到：(1) 上表并不全面，必须添加其他类以使模型更完整；(2) 某些被拒绝的潜在类将成为被接受类的属性（例如，编号和类型是 Sensor 的属性，主密码和电话号码可能成为 System 的属性）；(3) 对问题的不同陈述可能导致做出“接受或拒绝”的不同决定（例如，如果每个房主都有个人密码或通过声音确认，Homeowner 类就有可能接受并满足特征 1 和 2）。

10.2 描述属性

属性描述了已经选择包含在需求模型中的类。实质上，属性定义了类，以澄清类在问题空间的环境下意味着什么。例如，如果我们建立一个系统来跟踪职业棒球手的统计信息，那么类 Player 的属性与用于职业棒球手的养老系统中的属性就是截然不同的。前者，属性可能涉及名字、位置、平均击球次数、担任防守百分比、从业年限、比赛次数等相关信息。后者，以上某些属性仍是有意义的，但另外一些属性将会更换（或扩充），例如，平均工资、充分享受优惠权后的信用、所选的养老计划、邮件地址等。

关键点 属性是在问题的环境下完整定义类的数据对象集合。

为了给分析类开发一个有意义的属性集合，软件工程师应该研究用例并选择那些合理的“属于”类的“事物”。此外，每个类都应回答如下问题：什么数据项（组合项或基本项）能够在当前问题环境内完整地定义这个类？

188

为了说明这个问题，考虑为 SafeHome 定义 System 类。房主可以配置安全功能以反映传感器信息、报警响应信息、激活或者关闭信息、识别信息等。我们可以用如下方式表现这些组合数据项：

识别信息 = 系统编号 + 确认电话号码 + 系统状态

报警应答信息 = 延迟时间 + 电话号码

激活或者关闭信息 = 主密码 + 允许重试次数 + 临时密码

等式右边的每一个数据项可以进一步地细化到基础级，但是考虑到我们的目标，可以为 System 类组成一个合理的属性列表（图 10-1 中的阴影部分）。

传感器是整个 SafeHome 系统的一部分，但是并没有列出如图 10-1 所示的数据项或属性。已经定义 Sensor 为类，多个 Sensor 对象将和 System 类关联。通常，如果有超过一个项和某个类相关联，就应避免把这个项定义为属性。

10.3 定义操作

操作定义了某个对象的行为。尽管存在很多不同类型的操作，但通常可以粗略地划分为 4 种类型：(1) 以某种方式操作数据（例如添加、删除、重新格式化、选择）；(2) 执行计算的操作；(3) 请求某个对象的状态的操作；(4) 监视某个对象发生某个控制事件的操作。这些功能通过在属性或相关属性（10.5 节）上的操作来实现。因此，操作必须“理解”类的属性和相关属性的性质。

189

在第一次迭代要导出一组分析类的操作时，可以再次研究处理说明（或用例）并合理地选择属于该类的操作。为了实现这个目标，可以再次研究语法解析并分离动词。这些动词中的一部分将是合法的操作并能够很容

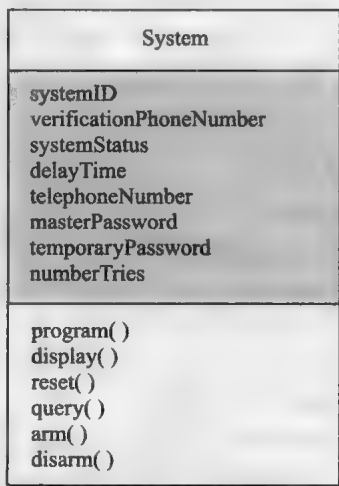


图 10-1 System 类的类图

建议 当为分析类定义操作时，集中于面向问题的行为而不是实现所要求的行为。

易地连接到某个特定类。例如，从本章前面提出的 SafeHome 处理说明中可以看到，“为传感器分配编号和类型”“主密码用于激活和解除系统”，这些短语表明：

- assign() 操作和 Sensor 类相关联。
- program() 操作应用于 System 类。
- arm() 和 disarm() 应用于 System 类。

再进一步研究，program() 操作很可能被划分为一些配置系统所需要的更具体的子操作。例如，program() 隐含着电话号码、配置系统特性（如创建传感器表、输入报警特征值）和输入密码。但是我们暂时把 program() 指定为一个单独的操作。

另外，对于语法解析，分析师能通过考虑对象间所发生的通信获得对其他操作的更为深入的了解。对象通过传递信息与另一个对象通信。在继续对操作进行说明之前，我们探测到了更详实的信息。

SafeHome 类模型

[场景] Ed 的小房间，开始进行分析建模。

[人物] Jamie、Vinod 和 Ed，SafeHome 软件工程团队的成员。

[对话]

Ed 已经从 ACS-DCV（本章前面的 SafeHome 中已有介绍）的用例模板中做了提取类方面的工作，并向他的同事展示了已经提取的类。

Ed：那么当房主希望选择一个摄像机的時候，他必须从一个平面设计图中进行选择。我已经定义了一个 FloorPlan 类，这个图在这里。

（他们查看图 10-2。）

Jamie：FloorPlan 这个类把墙、门、窗和摄像机都组织在一起。这就是那些标记线的意义，是吗？

Ed：是的，它们被称作“关联”，一个类根据我在图中所表示的关联关系和另一个类相关联（在 10.5 节中讨论关联）。

Vinod：那么实际的平面设计图是由墙构成的，并包含摄像机和放置在那些墙中的传感器。平面设计图如何知道在哪里放置那些对象？

Ed：平面设计图不知道，但是其他类知道。例如查看属性 WallSegment，该属性用于

构建墙，墙段（WallSegment）具有起点坐标和终点坐标，其他由 draw() 操作完成。

Jamie：这些也适用于门窗。看起来摄像机有一些额外的属性。

Ed：是的，我要求它们提供转动信息和缩放信息。

Vinod：我有个问题，为什么摄像机有 ID 编号而其他对象没有呢？我注意到有个属性叫 nextWall。WallSegment 如何知道什么是下一堵墙？

Ed：好问题，但正如他们所说，那是由设计决定的，所以我将推迟这个问题直到……

Jamie：让我休息一下……我打赌你已经想出办法了。

Ed（羞怯地笑了笑）：确实，当我们开始设计时我要采用列表结构来建模。如果你坚信分析和设计是分离的，那么我安排的详细程度等级就有问题了。

Jamie：对我而言看上去非常好。只是我还有一些问题。

（Jamie 问了一些问题，因此做了一些小的修改。）

Vinod：你有每个对象的 CRC 卡吗？如果有，我们应该进行角色演练，以确保没有

任何遗漏。

Vinod：这不难，而且确实有用，我给你演示一下。

Ed：我不太清楚如何做。

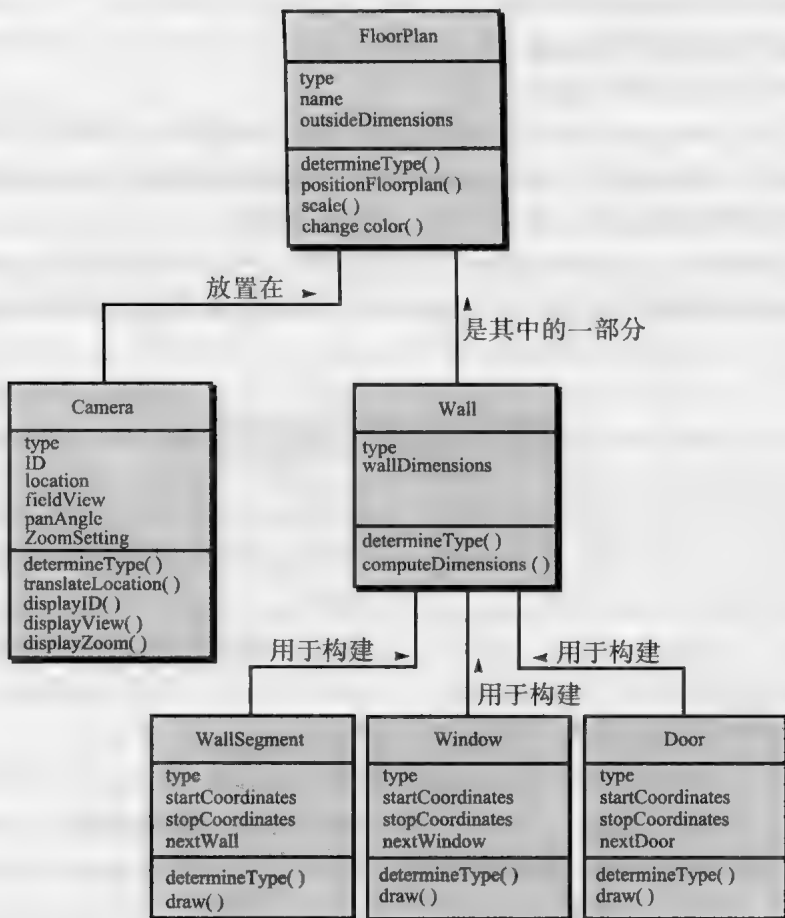


图 10-2 FloorPlan 类的类图

191

10.4 类 – 职责 – 协作者建模

类 – 职责 – 协作者 (Class-Responsibility-Collaborator, CRC) 建模 [Wir90] 提供了一个简单方法，可以识别和组织与系统或产品需求相关的类。Ambler[Amb95] 用如下文字解释了 CRC 建模：

CRC 模型实际上是表示类的标准索引卡片的集合。这些卡片分三部分，顶部写类名，卡片主体左侧部分列出类的职责，右侧部分列出类的协作者。

事实上，CRC 模型可以使用真实的或虚拟的索引卡，意图是有组织地表示类。职责是和类相关的属性和操作。简单地说，职责就是“类所知道或能做的任何事” [Amb95]。协作者是提供完成某个职责所需要信息的类。通常，协作意味着信息请求或某个动作请求。

FloorPlan 类的一个简单 CRC 索引卡如图 10-3 所示。CRC 卡上所列出的职责只是初步

引述 使用 CRC 卡的一个目的是尽早舍弃、频繁舍弃以及低成本舍弃。事实上抽出一叠卡片比改编大量源代码要容易得多。

C. Horstmann

的，可以添加或修改。在职责栏右边的 Wall 和 Camera 是需要协作的类。

类: FloorPlan	
说明	
职责:	协作者:
定义住宅平面图的名称 / 类型	
管理住宅平面图的布局	
缩放显示住宅平面图	
合并墙、门和窗	Wall
显示摄像机的位置	Camera

图 10-3 CRC 模型索引卡

类。在本章前面已经说明了识别类和对象的基本原则。10.1 节所说的类的分类可以通过如下分类方式进行扩展：

- 实体类，也称作模型或业务类，是从问题说明中直接提取出来的（例如 FloorPlan 和 Sensor）。这些类一般代表保存在数据库中和贯穿在应用程序中（除非被明确删除）的事物。
- 边界类用于创建用户可见的和在使用软件时交互的接口（如交互屏幕或打印的报表）。实体类包含对用户来说很重要的信息，但是并不显示这些信息。边界类的职责是管理实体对象呈现给用户的方式。例如，Camera Window 的边界类负责显示 SafeHome 系统监视摄像机的输出。
- 控制类自始至终管理“工作单元”。也就是说，控制类可以管理：
(1) 实体类的创建或更新；
(2) 边界类从实体对象获取信息后的实例化；
(3) 对象集合同的复杂通信；
(4) 对象间或用户和应用系统间交换数据的确认。通常，直到设计开始时才开始考虑控制类。

职责。在 10.2 节和 10.3 节中已经说明了识别职责（属性和操作）的基本原则。Wirfs-Brock 和她的同事 [Wir90] 在给类分配职责时建议了以下 5 个指导原则。

1. 智能系统应分布在所有类中以求最大程度地满足问题的需求。每个应用系统都包含一定程度的智能，也就是系统所知道的以及所能完成的。智能在类中可以有多种分布方式。建模时可以把“不灵巧”（Dumb）类（几乎没有职责的类）作为一些“灵巧”类（有很多职责的类）的从属。尽管该方法使得系统中的控制流简单易懂，但同时有如下缺点：把所有的智能集中在少数类，使得变更更为困难；将会需要更多的类，因此需要更多的开发工作。

如果智能系统更平均地分布在应用系统的所有类中，每个对象只了解和执行某些

网络资源 关于这些类的类型的精彩讨论可以参阅 <http://www.oracle.com/technetwork/develop-tools/jdev/gettingstartedwithumlclassmodeling-130316.pdf>。

引述 对象可以科学地分为三个主要类别：不工作的、损坏的和丢失的。

Russell Baker

提问 为类分配职责时可以采用什么指导原则？

事情（通常是适度集中），并提高系统的内聚性^①，这将提高软件的可维护性并减少变更的副作用。

193

为了确定分布智能系统是否恰当，应该评估每个 CRC 模型索引卡上标记的职责，以确定某个类是否应该具有超长的职责列表。如果有这种情况就表明智能太集中^②。此外，每个类的职责应表现在同一抽象层上。例如在聚合类 `CheckingAccount` 操作列表中，评审人员注意到两项职责：账户余额和已结算的支票。第一个操作的职责意味着复杂的算术和逻辑过程，第二个操作的职责是指简单的办事员活动。既然这两个操作不是相同的抽象级别，因此已结算的支票应该被放在 `CheckEntry` 的职责中，这是由聚合类 `CheckingAccount` 压缩得到的一个类。

2. **每个职责的说明应尽可能具有普遍性。**这条指导原则意味着应在类的层级结构的上层保持职责（属性和操作）的通用性（因为它们更有一般性，将适用于所有的子类）。
3. **信息和与之相关的行为应放在同一个类中。**这实现了面向对象原则中的封装，数据和操作数据的处理应包装在一个内聚单元中。
4. **某个事物的信息应局限于一个类中而不要分布在多个类中。**通常应由一个单独的类负责保存和操作某特定类型的信息。通常这个职责不应由多个类分担。如果信息是分布的，软件将变得更加难以维护，测试也会面临更多挑战。
5. **适合时，职责应由相关类共享。**很多情况下，各种相关对象必须在同一时间展示同样的行为。例如，考虑一个视频游戏，必须显示如下类：`Player`、`PlayerBody`、`PlayerArms`、`PlayerLegs` 和 `PlayerHead`。每个类都有各自的属性（例如 `position`、`orientation`、`color` 和 `speed`），并且所有属性都必须在用户操纵游戏杆时进行更新和显示。因此，每个对象必须共享职责 `update` 和 `display`。`Player` 知道在什么时候发生了某些变化并且需要 `update` 操作。它和其他对象协作获得新的位置或方向，但是每个对象控制各自的显示。

194

协作。类用一种或两种方法来实现其职责：（1）类可以使用其自身的操作控制各自的属性，从而实现特定的职责；（2）类可以和其他类协作。`Wirfs-Brock` 和她的同事 [Wir90] 这样定义协作：

协作是以客户职责实现的角度表现从客户到服务器的请求。协作是客户和服务器之间契约的具体实现……如果为了实现某个职责需要发送任何消息给另一个对象，我们就说这个对象和其他对象有协作。单独的协作是单向流，即表示从客户到服务器的请求。从客户的角度看，每个协作都和服务器的某个特定职责实现相关。

要识别协作可以通过确认类本身是否能够实现自身的每个职责。如果不能实现每个职责，那么需要和其他类交互，因此就要有协作。

例如，考虑 `SafeHome` 的安全功能。作为活动流程的一部分，`ControlPanel` 对象必须确定是否启动所有的传感器，定义名为 `determine-sensor-status()` 的职责。如果传感器是开启的，那么 `ControlPanel` 必须设置属性 `status` 为“未准备好”。传感器信息可以从每个 `Sensor` 对象获取，因此只有当 `ControlPanel` 和 `Sensor` 协作时才能实现 `determine-sensor-status()` 职责。

① 内聚性是一个设计概念，将在第 12 章中讨论。

② 在这种情况下，可能需要将一个类分成多个类或子系统，以便更有效地分布智能。

为帮助识别协作者，分析师可以检查类之间三种不同的通用关系 [Wir90]：(1) is-part-of (是……一部分) 关系；(2) has-knowledge-of (有……的知识) 关系；(3) depends-upon (依赖……) 关系。在下面的段落中将简单地分别说明这三种通用关系。

属于某个聚合类一部分的所有类可通过 is-part-of 关系和聚合类连接。考虑前面提到的视频游戏中所定义的类，PlayerBody 是 Player 的一部分，PlayerArms、PlayerLegs 和 PlayerHead 也类似。在 UML 中，使用如图 10-4 所示的聚合方式表示这些关系。

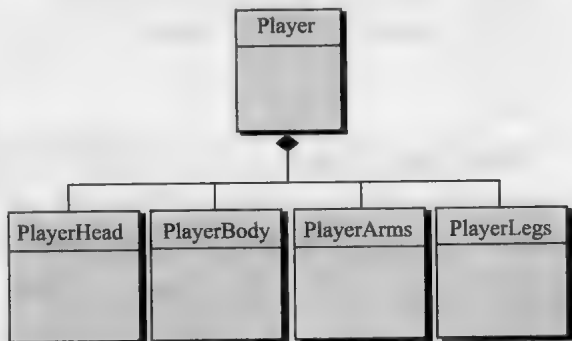


图 10-4 复合聚合类

当一个类必须从另一个类中获取信息时，就建立了 has-knowledge-of 关系。前面所说的 determine-sensor-status() 职责就是 has-knowledge-of 关系的一个例子。

depends-upon 关系意味着两个类之间具有 has-knowledge-of 和 is-part-of 不能实现的依赖关系。例如，PlayerHead 通常必须连接到 PlayerBody（除非视频游戏特别暴烈），然而每个对象并没有其他对象的直接信息。PlayerHead 对象的 center-position 属性由 PlayerBody 的中心位置确定，该信息通过第三方对象 Player 获得，即 PlayerBody 需要 Player。因此，PlayerHead 依赖 PlayerBody。

195

在所有情况下，我们都把协作类的名称记录在 CRC 模型索引卡上，紧靠在协作的职责旁边。因此，索引卡包含一个职责列表以及相关的能够实现这些职责的协作（图 10-3）。

当开发出一个完整的 CRC 模型时，利益相关者可以使用如下方法评审模型 [Amb95]。

1. 所有参加（CRC 模型）评审的人员拿到一部分 CRC 模型索引卡。拆分协作卡片（也就是说每个评审员不得有两张存在协作关系的卡片）。
2. 分类管理所有的用例场景（以及相关的用例图）。
3. 评审组长细致地阅读用例。当评审组长看到一个已命名的对象时，给拥有相应类索引卡的人员一个令牌。例如，SafeHome 的一个用例包含如下描述：

房主观察 SafeHome 控制面板以确定系统是否已经准备接收输入。如果系统没有准备好，房主必须手工关闭窗口（门）以便指示器呈现就绪状态。（未就绪指示器意味着某个传感器是开启的，也就是说某个门或窗户是打开的。）

当评审组长看到用例说明中的“控制面板”，就把令牌传给拥有 ControlPanel 索引卡的人员。“意味着某个传感器是开启的”语句需要索引卡包含确认该含义的职责（由 determine-sensor-status() 实现该职责）。靠近索引卡职责的是协作者 Sensor，然后令牌传给 Sensor 对象。

196

4. 当令牌传递时，该类卡的拥有者需要描述卡上记录的职责。评审组确定（一个或多个）职责是否满足用例需求。

5. 如果记录在索引卡上的职责和协作不能满足用例，就需要修改卡片。修改可能包括定义新类（和相关的 CRC 索引卡），或者在已有的卡上说明新的或修改的职责、协作。

该过程持续进行直到用例（或用例图）编写结束。评审完所有用例后将继续进行需求建模。

SafeHome CRC 模型

[场景] Ed 的办公室，刚开始需求建模。

[人物] Vinod 和 Ed，SafeHome 软件工程师团队成员。

[对话]

(Vinod 已经决定通过一个例子向 Ed 展示如何开发 CRC 卡。)

Vinod：在你着手于监视功能而 Jamie 忙着安全功能的时候，我在准备住宅管理功能。

Ed：情况怎样？市场营销人员的想法总是在变化。

Vinod：这是整个功能的第一版用例……我们已经改进了一点，它应该能提供一个整体视图。

用例：SafeHome 住宅管理功能。

说明：我们希望通过个人计算机上的住宅管理接口或互联网连接，来控制有无线接口控制器的电子设备。系统应该允许我打开或关闭指定的灯，控制连接到无线接口的设备，设置取暖和空调系统达到预定温度。为此，我希望从房屋平面图上选择设备。每个设备必须在平面图上标识出来。作为可选的特性，我希望控制所有的视听设备——音响设备、电视、DVD、数字录音机等。

通过不同选项就能够针对各种情况设置整个房屋，第一个选项是“在家”，第二个是“不在家”，第三个是“彻夜不归”，第四个是“长期外出”。所有这些情况都适用于所有设备的设置。在彻夜不归和长期外出时，系统将以随机的间隔时间开灯和关灯（以造成有人在家错觉），并控制取暖和空调系统。我应能够通过有适当密码保护的互联网撤销这些设置……

Ed：那些负责硬件的伙计已经设计出所有的无线接口了吗？

Vinod（微笑）：他们正在忙这个，据说没有问题。不管怎样，我从住宅管理中提取

了一批类，我们可以用一个做例子。就以 HomeManagementInterface 类为例吧！

Ed：好，那么职责是什么？类及协作者的属性和操作是那些职责所指向的类。

Vinod：我想你还不了解 CRC。

Ed：可能有点，但还是继续吧。

Vinod：这就是我给出的 HomeManagementInterface 的类定义。

属性：

optionsPanel——在按钮上提供信息，用户可以使用这些信息选择功能。

situationPanel——在按钮上提供信息，用户可以使用这些信息选择环境。

floorPlan——类似于监视对象，但这个用来显示设备。

deviceIcons——图标上的信息，代表灯、家用电器、HVAC 等。

devicePanels——模拟家用电器或设备控制面板，允许控制。

操作：

displayControl(), selectControl(), displaySituation(), selectSituation(), accessFloorplan(), selectDeviceIcon(), displayDevicePanel(), accessDevicePanel()……

类：HomeManagementInterface

职责	协作者
displayControl ()	OptionsPanel (类)
selectControl ()	OptionsPanel (类)
displaySituation ()	SituationPanel (类)
selectSituation ()	SituationPanel (类)
accessFloorplan ()	FloorPlan (类)
……	……

Ed：这样，当调用 accessFloorPlan() 操作时，将和 FloorPlan 对象协作，类似我们为监视开发的对象。等一下，我这里有它的说明（他们查看图 10-2）。

Vinod：确实如此。如果我们希望评审整个类模型，可以从这个索引卡开始，然后

到协作者的索引卡，再到协作者的协作者的索引卡，依此类推。

Ed: 这真是个发现遗漏和错误的好方法。

Vinod: 的确如此。

10.5 关联和依赖

在很多例子中，两个分析类以某种方式相互联系着。在 UML 中，这些联系被称作关联（accociation）。参考图 10-2，通过识别 FloorPlan 与另外两个类 Camera 和 Wall 之间的一组关联确定 FloorPlan 类。类 Wall 和三个构成墙类 WallSegment、Window 和 Door 相关联。

关键点 关联定义了类之间的联系，多样性定义了一个类和另一个类之间联系的数量关系。

在某些情况下，关联可以更进一步地指出多样性。参考图 10-2，一个 Wall 对象可以由一个或多个 WallSegment 对象构成。此外，Wall 对象可能包含 0 或多个 Window 对象以及 0 或多个 Door 对象。这些多样性限制如图 10-5 所示，其中“一个或多个”使用 1..* 表示，“0 或多个”使用 0..* 表示。在 UML 中，星号表示范围无上界。⊖

在很多事例中，两个分析类之间存在客户-服务器关系。这种情况下，客户类以某种方式依赖于服务器类并且建立了依赖关系。依赖性是由一个构造型（stereotype）定义的。在 UML 中，构造型是一个“可扩展机制”[Arl02]，允许软件工程师定义特殊的建模元素，这些建模元素的语义是由用户自定义的。在 UML 中，构造型由一对尖括号表示（如 <<stereotype>>）。

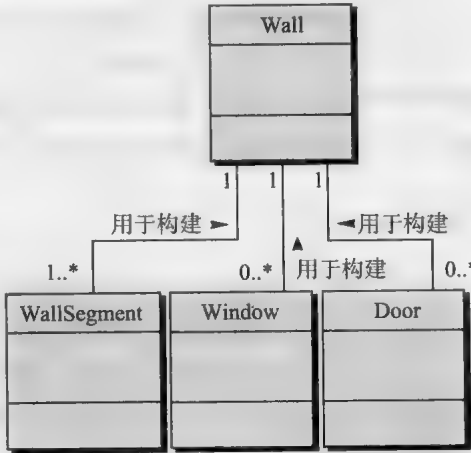


图 10-5 多样性

下面举例说明 SafeHome 监视系统中的简单依赖关系。Camera 对象（本例中的服务器类）向 DisplayWindow 对象（本例中的客户类）提供视频图像。这两个对象之间的关系不是简单的关联，而是存在着依赖关系。在监视用例中（没有列出来），建模者知道必须提供特定的密码才能查看指定摄像机的位置。其实现的一种方法是让 Camera 请求密码，然后在获得 DisplayWindow 的允许后显示视频。这可以由图 10-6 表示，其中 <<access>> 意味着通过特定的密码控制对摄像机输出的使用。

提问 什么是构造型？

198



图 10-6 依赖性

10.6 分析包

分析建模的一部分重要工作是分类，也就是将分析模型的各种元素（如用例、分析类）以一种方式分类，分组打包后称为分析包，并取一个有代表性的名字。

关键点 分析包用来集合一组相关的类。

⊖ 其他的多样性关联（一对一、一对多、多对多、一对某指定上下限的范围，以及其他）可以标识为关联的一部分。

199

为了说明分析包的使用，考虑我们前面所说的视频游戏。假设视频游戏的分析模型已经建成，并且已经生成大量的类。有一些类关注游戏环境，即用户游戏时所看到的可视场景。Tree、Landscape、Road、Wall、Bridge、Building、VisualEffect 类等可能就属于游戏环境类。另一些类关注游戏内的人物，说明他们的肢体特点、动作和局限。也可能定义了（前面提到的）Player、Protagonist、Antagonist、SupportingRoles 类。还需要一些类来说明游戏的规则，即游戏者如何穿越环境。例如这里可以选 RulesOfMovement 类和 ConstraintsOnAction 类。还可能存在很多其他类，可以用图 10-7 中的分析包表示这些类。

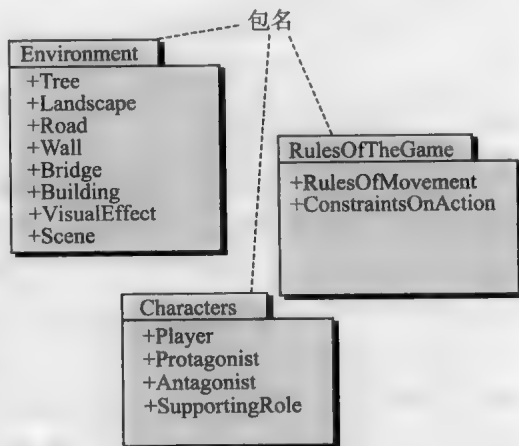


图 10-7 包

每个包中分析类名字前的加号表示该类是公共可见的，因此可以从其他包访问。尽管目前的图中没有显示，但包中的任何元素之前都可以添加其他符号。负号表示该元素对其他包是隐藏的，# 号表示只能由指定包中的类访问该元素。

10.7 小结

为了识别分析类，基于类的建模使用从用例和其他编写的应用描述中导出的信息。可以使用语法解析从文本说明中提取候选类、属性和操作，并制定用于定义类的标准。

200

CRC 索引卡可以用于定义类之间的联系。此外，可以使用各种 UML 建模方法定义类之间的层次、关系、关联、聚合和依赖。使用分析包方式进行分类和分组，在某种意义上为大型系统提供了更好的管理。

习题与思考题

10.1 构建如下系统中的一个：

- 你所在大学中基于网络的课程注册系统。
- 一个计算机商店的基于 Web 的订单处理系统。
- 一个小企业的简单发票系统。
- 内置于电磁灶或微波炉的互联网食谱。

选择你感兴趣的系统开发一套处理说明。然后使用语法解析技术识别候选对象和类。

10.2 使用问题 10.1 中识别的类开发一系列操作。

10.3 为问题 9.5 中表述的 PHTRS 系统开发一个类模型。

10.4 为 10.4 节中非正式描述的 SafeHome 住宅管理系统编写一个基于模板的用例。

10.5 为问题 10.1 所选的产品或系统开发一组完整的 CRC 模型索引卡。

10.6 指导你的同事一起评审 CRC 索引卡，评审结果中增加了多少类、职责和协作者？

10.7 什么是分析包？如何使用分析包？

扩展阅读与信息资源

Weisfeld 讨论了常规的基于类的观点（《The Object-Oriented Thought Process》，4th ed., Addison-Wesley, 2013）。讨论基本类的建模方法的书包括：Bennet 和 Farmer（《Object-Oriented Systems

Analysis and Design Using UML 》, McGraw-Hill, 2010), Ashrafi (《 Object-Oriented Systems Analysis and Design 》, Prentice Hall, 2008), Booch (《 Object-Oriented Analysis and Design with Applications 》, 3rd ed., Addison-Wesley, 2007), George 和他的同事 (《 Object-Oriented Systems Analysis and Design 》, 2nd ed., Prentice Hall, 2006), O'Docherty (《 Object-Oriented Analysis and Design 》, Wiley, 2005), Satzinger 等人 (《 Object-Oriented Analysis and Design with the Unified Process 》, Course Techonlogy, 2004), Stumpf 和 Teague (《 Object-Oriented Systems Analysis and Design with UML 》, Prentice Hall, 2004)。

UML 建模技术可以应用于分析和设计, 讨论这方面的书包括: Dennis 和他的同事 (《 Systems Analysis and Design with UML Version 2.0 》, Wiley, 4th ed., 2012), Ramnath 和 Dathan (《 Object-Oriented Analysis and Design 》, Springer, 2011), Bennett 和 Farmer (《 Object-Oriented System Analysis and Design Using UML 》, McGraw-Hill, 4th ed., 2010), Larman (《 Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development 》, Dohrling Kindersley, 2008), Rosenberg 和 Stephens (《 Use Case Driven Object Modeling with UML Theory and Practice 》, Apress, 2007), 还有 Arlow 和 Neustadt (《 UML 2 and the Unified Process 》, 2nd ed., Addison-Wesley, 2005)。这些书都描述了用 UML 内容分析类的定义。

网上有很多关于基于类的方法进行需求建模的信息。可以参考 SEPA 网站 www.mhhe.com/pressman 上有关分析建模的最新参考文献。

需求建模：行为、模式和 Web / 移动 App

要点浏览

概念：在这一章将学习需求建模的其他维度——面向行为建模、模式，以及开发 WebApp 时要考虑的特定需求分析问题。这里的每种模型都是第 9 章和第 10 章的补充。

人员：软件工程师（有时称为分析师）从各个利益相关者中提取需求进行建模。

重要性：软件工程师洞悉到软件需求的增长与来自不同需求模型维度的增长成正比。尽管可能没有时间，缺乏资源，无法采用第 9 ~ 11 章所建议的任何一种表示方法进行开发，但是软件工程师有必要认知每种不同的建模方式，这会给他提供审视问题的不同方法。继而他（和其他利益相关者）将能够更好地进入状态，

更好地界定是否已把必须完成的需求真正描述清楚。

步骤：行为建模描述了系统及其类的状态，以及事件对这些类的影响。基于模式的建模利用现有领域的知识使得需求分析更为容易。WebApp 的需求模型特别适用于表述内容、交互操作、功能和配置相关的需求。

工作产品：可为需求建模选择大量不同的基于文本和图形的表示方法。每种表示方法都提供了一种或多种模型元素。

质量保证措施：必须评审需求建模产品的正确性、完备性和一致性。必须反映所有利益相关者的要求，为导出设计建立基础。

在第 9 章和第 10 章讨论了基于场景建模和基于类模型后，我们很自然会问：“这些需求建模表示方法就足够了吗？”

唯一合理的回答是：“要看情况而定。”

对于某种类型的软件，用例可能是唯一可行的需求建模表示方法。而对于其他类型的软件，则需要选择面向对象的方法开发基于类的模型。但在另外一些情形下，可能必须要对复杂应用软件需求做个检测：查看当数据对象在系统中移动时是如何转换的；查看作为外部事件后果的应用软件是如何工作的；查看现存知识领域能否解决当前问题；或者在基于 Web 的系统或移动系统和应用软件中，如何将内容和功能融合在一起，使得最终用户能够利用 WebApp 实现相关目标。

关键概念

分析模式
行为模型
配置模型
内容模型
事件
功能模型
交互模型
导航建模
顺序图
状态图
状态表达

11.1 生成行为模型

前面章节讨论的建模表示方法表达了需求模型中的静态元素，现在则是把它转换成系统或产品的动态行为的时候了。要实现这一任务，可以把系统行为表示成一个特定的事件和时间的函数。

行为模型显示了软件如何对外部事件或激励做出响应。要生成模型，分析师必须按如下步骤进行：（1）评估所有的用例，以保证完全理解系统内的交互顺序；（2）识别驱动交互顺序的事件，并理解这些事件如何与特定的对象相互关联；（3）为每个用例生成序列；（4）创建系统状态图；（5）评审行为模型以验证准确性和一致性。

在下面的几节中将讨论每个步骤。

提问 如何用模型表明软件对某些外部事件的影响？

11.2 识别用例事件

在第 9 章中，我们知道用例表现了涉及参与者和系统的活动顺序。一般而言，只要系统和参与者之间交换了信息就发生事件。事件应该不是被交换的信息，而是已交换信息的事实。

为了说明如何从信息交换的角度检查用例，让我们再来考察 SafeHome 安全功能中的一部分用例。

房主使用键盘键入 4 位密码。系统将该密码与已保存的有效密码相比较。如果密码不正确，控制面板将鸣叫一声并复位以等待下一次输入；如果密码正确，控制面板等待进一步的操作。

用例场景中加下划线的部分表示事件。应确认每个事件的参与者，应标记交换的所有信息，而且应列出任何条件或限制。

举个典型事件的例子，考虑用例中加下划线的“房主使用键盘键入 4 位密码”。在需求模型的环境下，对象 Homeowner[⊖]向对象 ControlPanel 发送一个事件。这个事件可以称作输入密码。传输的信息是组成密码的 4 位数字，但这不是行为模型的本质部分。重要的是注意到某些事件对用例的控制流有明显影响，而其他事件对控制流没有直接影响。例如，事件输入密码不会明显地改变用例的控制流，但是事件比较密码（从与事件“系统将该密码与保存的有效密码相比较”的交互中得到）的结果将明显影响到 SafeHome 软件的信息流和控制流。

一旦确定了所有的事件，这些事件将被分配到所涉及的对象，对象负责生成事件（例如，Homeowner 房主生成输入密码事件）或识别已经在其他地方发生的事件（例如，ControlPanel 控制面板识别比较密码事件的二元结果）。

11.3 状态表达

在行为建模中，必须考虑两种不同的状态描述：（1）系统执行其功能时每个类的状态；（2）系统执行其功能时从外部观察到的系统状态。

类状态具有被动和主动两种特征 [Cha93]。被动状态只是某个对象所有属性的当前状态。例如，类 Player 的被动状态（在第 10 章讨论的视频游戏应用程序中）将包含 Player 当前的 position 和 orientation 属性，以及和游戏相关的 Player 的其他特性（例如，用来显示 magic wishes remaining 的属性）。对象的主动状态指的是对象进行持续变换或处理时的当前状态。

类 Player 可能具有如下的主动状态：移动、休息、受伤、疗伤、被捕、失踪等。事件（有时称为触发器）才能迫使对象做出从一个主动状态到另一个主动状态的转移。

关键点 系统状态可以表现特定的外部可观察的行为，类的状态可以表现当前系统执行功能时的行为。

[⊖] 在这个例子中，我们假设每位与 SafeHome 交互的用户（房主）都拥有确认的密码，因此都是合法的对象。

下面将讨论两种不同的行为表现形式。第一种显示一个类如何改变基于外部事件的状态，第二种以时间函数的形式显示软件的行为。

分析类的状态图。UML 状态图[⊖]就是一种行为模型，该图为每个类呈现了主动状态和导致这些主动状态发生变化的事件（触发器）。图 11-1 举例说明了 SafeHome 安全功能中 ControlPanel 类的状态图。

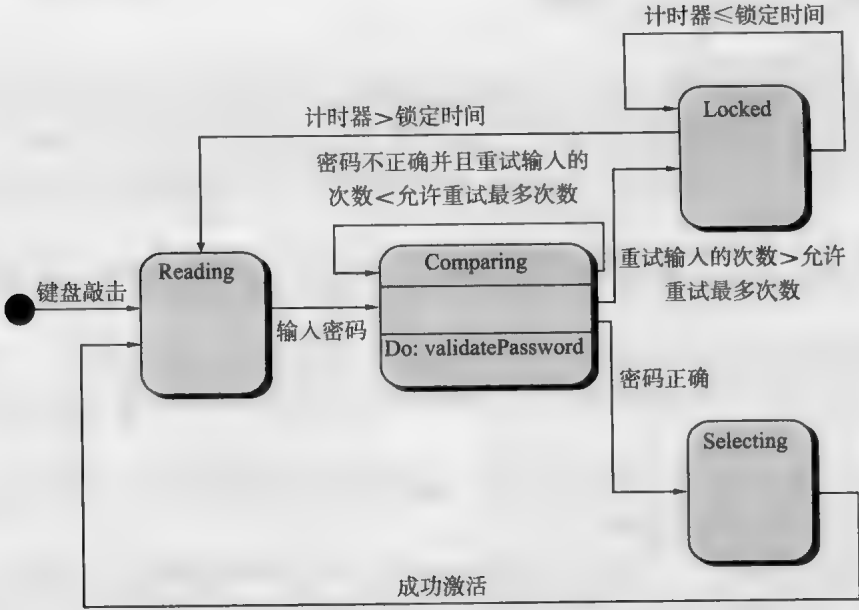


图 11-1 ControlPanel 类的状态图

图 11-1 中显示的每个箭头表示某个对象从一个主动状态转移到另一个主动状态。每个箭头上的标注都体现了触发状态转移的事件。尽管主动状态模型在提供对象的“生命历史”信息方面非常有用，但也能提供另外一些信息以便更深入地理解对象的行为。除了说明导致转移发生的事件外，分析师还可以说明守卫和动作 [Cha93]。守卫是为了保证转移发生而必须满足的一个布尔条件。例如，图 11-1 中从“读取”状态转移到“比较”状态的守卫可以由检查用例来确定：

if (password input = 4 digits) then compare to stored password

一般而言，转移的守卫通常依赖于某个对象的一个或多个属性值。换句话说，守卫依赖于对象的被动状态。

动作是与状态转移同时发生的，或者作为状态转移的结果，通常包含对象的一个或多个操作（职责）。例如，和输入密码事件（见图 11-1）相关联的动作是由 validatePassword() 操作的，该操作访问 password 对象并通过执行按位比较来验证输入的密码。

顺序图。第二种表现行为的方式在 UML 中称作顺序图，它表明事件如何引发从一个对象到另一个对象的转移。一旦通过检查用例确认了事件，建模人员就创建了一个顺序图，即用时间函数表现如何引发事件从一个对象流到另一个对象。事实上，顺序图是用例的速记版本。它表现了导致行为从一个类流

关键点 与不注明相关类表现行为的状态图不同，顺序图通过说明类如何从一个状态转移到另一状态来表现行为。

⊖ 如果读者对 UML 不熟悉，在附录 1 中有这些重要建模符号的简单介绍。

到另一个类的关键类和事件。

图 11-2 给出了 SafeHome 安全功能的部分顺序图。每个箭头代表了一个事件（源自一个用例）并说明了事件如何引导 SafeHome 对象之间的行为。时间纵向（向下）度量，窄的纵向矩形表示处理某个活动所用的时间。沿着纵向的时间线可以展示出对象的状态。

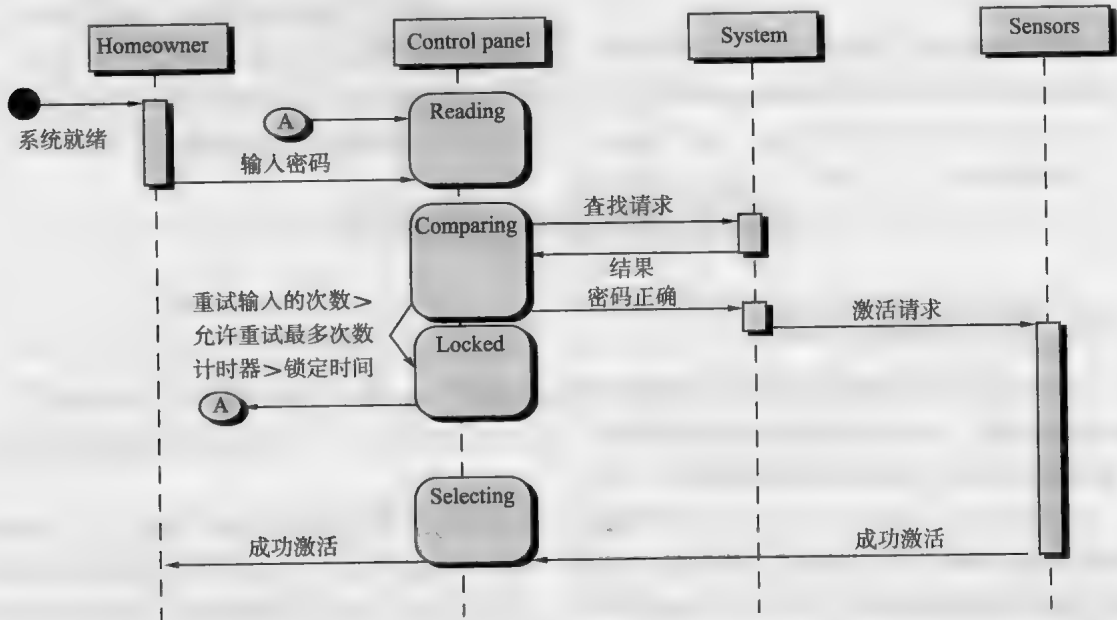


图 11-2 SafeHome 安全功能的顺序图（部分）

第一个事件系统就绪来自于外部环境并且把行为引导到对象 Homeowner。房主输入一个密码，查找请求事件发送到 System，它在一个简单的数据库中查找密码并向 ControlPanel（现在处在比较状态）返回（找到或没有找到）结果。有效的密码将促使系统产生密码正确事件，该事件通过激活请求事件激活传感器。最终，通过控制把成功激活事件返回到房主。

一旦完成了完整的顺序图，就把所有导致系统对象之间转移的事件整理为输入事件集合和输出事件集合（来自一个对象）。对于将要构建的系统而言，这些信息对于创建系统的有效设计非常有用。

206

软件工具 UML 中的通用分析建模

[目标] 分析建模工具可以使用 UML 语言开发基于场景的模型、基于类的模型和行为模型。

[机制] 这类工具支持构建分析模型所需的全部 UML 图（这些工具也支持设计建模）。除了制图，这类工具：（1）为所有的 UML 图执行一致性和正确性检查；

（2）为设计和代码生成提供链接；（3）构建数据库，以便实现管理和评估复杂系统所需的大型 UML 模型。

[代表性工具][⊖]

如下工具支持分析建模所需的全部 UML 图：

- ArgoUML。开源工具（argouml.tigris。

⊖ 这里记录的工具只是此类工具的例子，并不代表本书支持采用这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

- org)。
- Enterprise Architect。由 Sparx Systems 开发 (www.sparxsystems.com.au)。
- PowerDesigner。由 Sybase 开发 (www.sybase.com)。
- Rational Rose。由 IBM (Rational) 开发 (www-01.ibm.com/software/rational)。
- Rational System Architect。由 Popkin Software 开发, 现在归属于 IBM (<http://www-01.ibm.com/software/awdtools/systemarchitect/>)。
- UML Studio。由 Pragsoft Corporation 开发 (www.pragsoft.com)。
- Visio。由 Microsoft 开发 (<http://office.microsoft.com/en-gb/visio/>)。
- Visual UML。由 Visual Object Modelers 开发 (www.visualuml.com)。

11.4 需求建模的模式

软件模式是获取领域知识的一种机制, 在遇到新问题时可以反复使用。在某些情况下, 领域知识在同一应用领域中用于解决新问题。在另外一些情况下, 通过模式获取的领域知识可借助模拟用于完全不同的应用领域。

分析模式的最初创作者没有“创建”模式, 但在需求工程工作中发现了模式。一旦发现模式则记载“明确的常见问题: 哪种模式适用, 规定的解决方案, 在实践中使用模式的假设和约束, 以及关于模式的某些常见的其他消息, 如使用模式的动机和驱动力, 讨论模式的优缺点, 参考在实践应用中某些已知的使用模式的样例” [Dev01]。

[207]

第 8 章引入了分析模式的概念并指明这些模式表示了某些应用领域 (例如一个类、一个功能或一个行为), 当为这个领域的应用执行需求建模时会重复使用这些模式^①。分析模式都存储在一个仓库中, 以便软件团队的成员能够使用搜索工具找到并复用。一旦选择到合适的模式, 就可以通过引用模式名称将其整合到需求模型中。

11.4.1 发现分析模式

需求模型由各种元素组成: 基于场景 (用例)、基于类 (对象和类) 和行为 (事件和状态)。其中每个元素都是从不同的视角检查问题, 并且每一个都提供一种发现模式的机会, 可能发生在整个应用领域, 或者发生在类似但不同的应用领域。

在需求模型的最基本的元素是用例。在这个讨论环境下, 一套连贯用例可以成为服务于发现一个或多个分析模式的基础。语义分析模式 (Semantic Analysis Pattern, SAP) “描述了一小套连贯用例, 这些用例一起描述了通用应用的基础” [Fer00]。

下面我们考虑要求控制和监控“实时摄像机”以及汽车临近传感器的软件用例。

用例: 监控反向运动。

描述: 当车辆安装了反向装置后, 控制软件就能将后向摄像机的视频输入仪表板显示器。控制软件在仪表板显示器上叠加各种距离线和方向线, 以便车辆向后运动时驾驶员能保持方向。控制软件还能监控临近传感器, 以判定在车后方 10 英尺内是否有物体存在。如果临近传感器检测到某个物体在车后方 x 英尺内就会让车自动停止, 这个 x 值由车辆的速度决定。

^① 在软件设计中, 关于模式的进一步讨论见第 16 章。

在需求收集和建模阶段，本用例包含（在一套连贯用例中）各种将要精炼和详细阐述的功能。无论完成得如何精炼，建议用例要简单，但还要广泛地适用于 SAP，即具有基于软件的监控和在一个物理系统中对传感器和执行器的控制。在本例中，“传感器”提供临近信息和视频信息。“执行器”用于车辆的停止系统（如果一个物体离车辆很近就会调用它）。但是更常见的情况是发现大量的应用模式。

208

许多不同应用领域的软件需要监控传感器和控制物理执行器。所依照的分析模式描述了能广泛应用的通用需求。这个模式叫做 Actuator-Sensor（执行器-传感器），将用作 SafeHome 的部分需求模型，将在随后的 11.4.2 节讨论。

SafeHome 发现分析模式

[场景] 一个会议室，一个团队会议。

[人物] Jamie Lazar，软件团队成员；Ed Robbins，软件团队成员；Doug Miller，软件工程经理。

[对话]

Doug: SafeHome 项目有关传感器网络的需求建模进展如何啊？

Jamie: 对我来说传感器的工作有点新，但我认为我能掌控好。

Doug: 我们能帮上你什么忙吗？

Jamie: 如果我曾建立过类似的系统，会更容易些。

Doug: 当然。

Ed: 我考虑到在这种情况下我们要找到一个分析模式，帮助我们为这些需求建模。

Doug: 如果我们能发现正确的模式，我们可以避免重新发明轮子。

Jamie: 对我而言这太好了。那么我们怎么开始呢？

Ed: 我们有一个库包含大量的分析和设计模式。我们只需要检索出符合需求的模式。

Doug: 看来就这么干吧。你认为怎样，Jamie？

Jamie: 如果 Ed 能帮我开始，我今天就按此行动了。

11.4.2 需求模式举例：执行器-传感器^①

SafeHome 安全功能需求之一是能监控安全传感器（例如，入侵传感器，火、烟或一氧化碳传感器，水传感器）。基于互联网的 SafeHome 还将要求能控制住所内安全摄像机的活动（例如拍摄、变焦）。这意味着 SafeHome 软件必须管理各种传感器和“执行器”（例如摄像机控制机制）。

Konrad 和 Cheng[Kon02] 建议将需求模式命名为执行器-传感器，它为 SafeHome 软件这项需求的建模提供有用的指导。执行器-传感器模式的简约版本最初是为如下的汽车应用开发的。

模式名：执行器-传感器。

目的：详细说明在嵌入系统中的各种传感器和执行器。

动机：嵌入系统包含各种传感器和执行器，这些传感器和执行器都直接或间接连接到一个控制单元。虽然许多传感器和执行器看上去完全不同，但它们的行为是相似的，足以构成

209

① 本节采用 [Kon02] 的内容，已得到作者许可。

一个模式。这种模式显示了如何为系统指定传感器和执行器，包括属性和操作。执行器-传感器模式为被动式传感器使用拉机制（对消息的显示要求），为主动传感器使用推机制（消息广播）。

约束：

- 每个被动传感器必须通过某种方法来读取传感器的输入和表示传感器值的属性。
- 每个主动传感器必须能在其值发生变更时广播更新消息。
- 每个主动传感器应该能发送一个生命刻度，即在特定时间帧中发布状态信息，以便检测出错误动作。
- 每个执行器必须通过某种方法来调用由 ComputingComponent 计算构件决定的适当应答。
- 每个传感器和执行器应有实施检测其自身操作状态的功能。
- 每个传感器和执行器应能测试接收值或发送值的有效性，并且当值超出指定边界时能设定其操作状态。

适用性：对有多个传感器和执行器的任何系统都是非常有用的。

结构体：图 11-3 显示了执行器-传感器模式的 UML 类图，执行器、被动传感器和主动传感器是抽象类。这个模式中有四种不同的传感器和执行器。

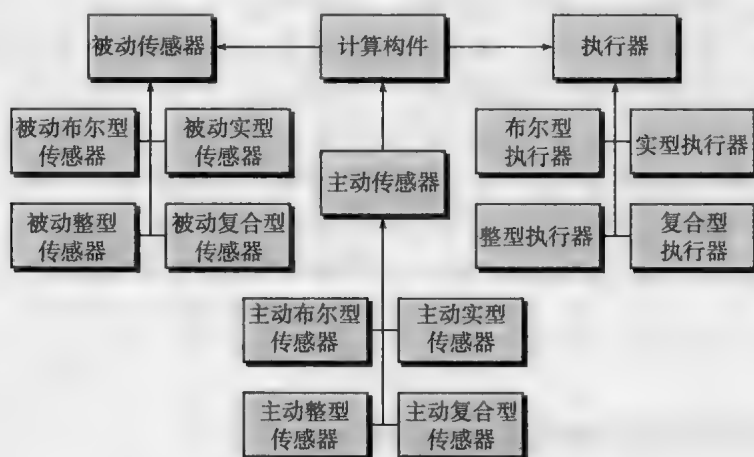


图 11-3 传感器和执行器的 UML 顺序图

传感器和执行器的常见类型为 Boolean、Integer 和 Real。就基本数据类型而言，复杂类是不能用值简单表示的，例如雷达设备。虽然如此，这些设备还是应该继承来自抽象类的接口，因为它们应该具备基本的功能（如查询操作状态）。

行为：图 11-4 描述了执行器-传感器例子的 UML 顺序图。这个例子可以应用于 SafeHome 功能，用以控制安全摄像机的调整（例如摇摄、变焦）。在读取或设置值时，ControlPanel 需要一个传感器（被动传感器）和一个执行器（摇动控制器）来进行以诊断为目的的操作状态核查。名为 Set Physical Value 和 Get Physical Value 的消息不是对象间的信息，相反，它们描述了系统物理设备和相关软件对应项之间的交互活动。在图的下部（即在水平线以下），PositionSensor 报告操作状态为零。接着 ComputingComponent 发送位置传感器失败的错误代码给 FaultHandler 错误处理程序，它将决定这些错误对系统的影响，以及需要采取的措施。从传感器获得的数据经过计算得到执行器所需的应答。

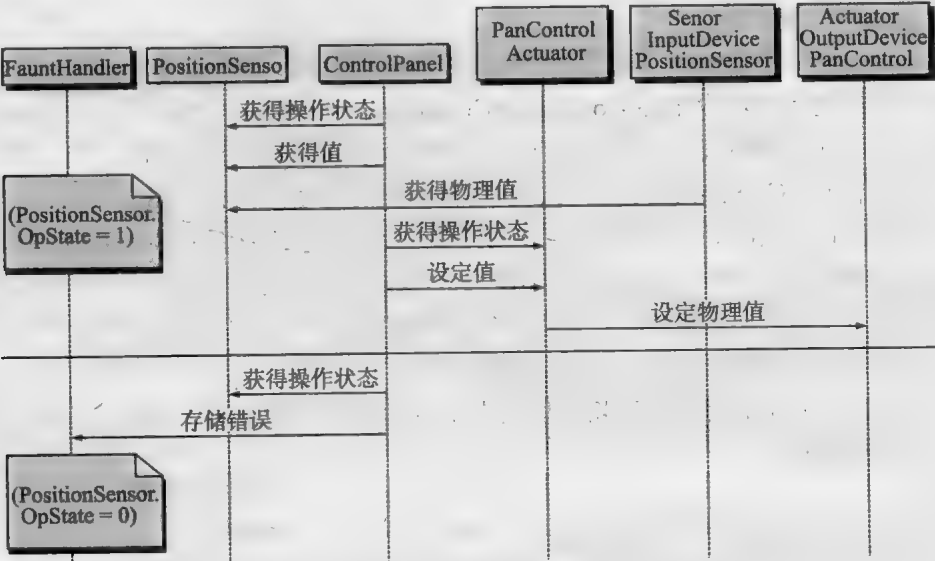


图 11-4 执行器 - 传感器模式的 UML 类图

210
211

参与者：模式“详细列举了包含在需求模式中的类或对象” [Kon02]，并描述了每个类或对象（图 11-3）的职责。简单列表如下：

- **PassiveSensor abstract：**定义被动传感器接口。
- **PassiveBooleanSensor：**定义被动布尔型传感器。
- **PassiveIntegerSensor：**定义被动整型传感器。
- **PassiveRealSensor：**定义被动实型传感器。
- **ActiveSensor abstract：**定义主动传感器接口。
- **ActiveBooleanSensor：**定义主动布尔型传感器。
- **ActiveIntegerSensor：**定义主动整型传感器。
- **ActiveRealSensor：**定义主动实型传感器。
- **Actuator abstract：**定义执行器接口。
- **BooleanActuator：**定义布尔型执行器。
- **IntegerActuator：**定义整型执行器。
- **RealActuator：**定义实型执行器。
- **ComputingComponent：**控制器的中心部分，它从传感器得到数据并为执行器计算所需的应答。
- **ActiveComplexSensor：**主动复合型传感器具备抽象类 **ActiveSensor** 的基本功能，但还需要指定额外更详细的方法和属性。
- **PassiveComplexSensor：**被动复合型传感器具备抽象类 **PassiveSensor** 的基本功能，但还需要指定额外更详细的方法和属性。
- **ComplexActuator：**复合型执行器具备抽象类 **Actuator** 的基本功能，但还需要指定额外更详细的方法和属性。

协作。本节描述的是对象和类之间如何进行交互活动，以及它们各自如何实现自身的责任。

- 当 **ComputingComponent** 需要更新 **PassiveSensor** 的值时，它询问传感器，通过发送

适当的消息请求传值。

- **ActiveSensor** 无需查询。这些对象和类启动传送过程，向计算机单元中传送传感器的值，使用适当方法设定在 **ComputingComponent** 中的值。对象和类在指定的时间帧发送至少一次生命刻度，以便使用系统的时钟时间更新它们的时间戳。
- 当 **ComputingComponent** 需要设定执行器的值时，便会给执行器发送值。
- **ComputingComponent** 能使用适当的方法查询和设定传感器和执行器的操作状态。如果发现操作状态为零，就发送错误给 **FaultHandler** 错误处理程序，这个类包含处理错误消息的方法，比如启动更详细的恢复机制或者一个备份设备。如果不可恢复，系统只能使用传感器最后的已知值或者默认值。
- **ActiveSensor** 提供增加或消除地址或者组件地址范围的方法，一旦值发生变化，组件就能获得消息。

结果：

1. 传感器和执行器类有一个通用接口。
2. 只能通过消息访问类的属性，并且类决定是否接受这个消息。例如，如果执行器的值超过其最大值，执行器类就不可能接收消息，或者使用默认的最大值。
3. 系统潜在的复杂度得以降低是因为传感器和执行器有统一的接口。

需求模式的描述也能为其他相关的需求模式和设计模式提供参考。

11.5 Web / 移动 App 的需求建模^①

Web/ 移动 App 开发者时常质疑我们所建议的需求分析观念。他们争辩道：“开发过程终归必须使用敏捷方法，并且分析是非常耗时的。当我们需要设计和建立系统时就会减慢工作进度。”

需求分析确实很花时间，但是解决错误的问题甚至更花时间。每个 WebApp 和移动开发者的问题都不复杂，但是能否确保理解了问题或产品的需求？如果答案是毫不含糊的“是”，那么可以跳过需求建模，但是如果答案是“否”，就应该实施需求建模。

11.5.1 多少分析才够用

Web/ 移动 App 需求建模的重要程度依据以下因素而定：（1）应用软件的规模和复杂度的差异；（2）利益相关者的人数（所做的分析能帮助识别不同来源的需求冲突）；（3）WebApp 团队的人数；（4）一起工作以前该开发团队成员的业务水平（所做的分析有助于达成对项目的共同理解）；（5）组织成功的程度直接依赖于应用软件的成功。

与前面相反的观点是，当项目规模更小、利益相关者更少、开发团队更有内聚力，同时应用软件并非十分重要时，采用更轻量级的分析方法是合理的。

虽然在开始设计前进行问题或产品需求的分析是个好主意，但并不见得所有分析一定在所有设计之前。事实上，应用软件中仅有特定部分的设计要求分析对应用软件有影响的需求。例如 SafeHome，对所有网站做符合要求的美学设计（版面布局、色彩框架等）不需要分析电子商务能力的功能需求。为提高交付成果，软件分析师只需要分析与递增式交付相关

① 本节部分内容来自 Pressman 和 Lowe[Pre08]，已得到作者许可。

的设计工作部分。^①

11.5.2 需求建模的输入

当 Web/移动 App 已经工程化时，可以采用第 5 章讨论的常规软件过程的敏捷版本。这个过程包含的沟通活动可以识别利益相关者和用户类别、业务环境、界定信息和可用目标、普通的 WebApp 需求和使用场景，这些都是能成为需求建模输入的信息。这些信息以自然语言的描述、概要大纲、草图以及其他非正式形式表达。

分析这类信息，并使用（所适用的）正式定义的表达模式构造它，接着生成更缜密的模型作为输出。需求模型提供了揭示问题真实结构的详细指导，并提供洞察其特性的解决方案。

在第 9 章介绍 SafeHome 的 ACS-DCV（摄像机监视）功能时这个功能看上去相当清晰，并描述了用例的部分细节（9.2.1 节）。然而，当再次审查用例时就可能发现丢失的信息、模棱两可的信息或不清晰的信息。

某方面的丢失信息自然会在设计阶段显露出来。例如功能键的特定布局、美学外观、快照视图尺寸、摄像机观察点位置、房间地板平面图，或者如密码的最大和最小长度这些细节。这些方面的某些内容是由设计决定的（如按键布局），另一些内容是根本不能影响早期设计工作的需求（如密码长度）。

但是有些丢失的信息实际上可能会影响总体设计，更多的则与需求的实际理解相关。例如：

问题 1：通过 SafeHome 摄像机输出的视频分辨率是多少？

问题 2：如果正在被监控的摄像机遇到警报条件，会发生什么？

问题 3：系统如何操作能摇摄和变焦的摄像机？

问题 4：摄像机的视图上还应该提供哪些信息？（例如，位置？时间或日期？上一次访问？）

在用例的初始开发阶段都没有考虑到这些问题，但是这些问题的答案却对设计的不同方面有着很大影响。

虽然沟通活动提供了很好的理解基础，但是需求分析通过提供额外的解释会使理解更精确，因此这个结论是合理的。当把问题的结构描述成需求模型的部分内容时，新问题又出现了。这就是弥补理解差距的问题，（或者在某些情况下）实际上这些问题也帮助我们在第一时间发现了这些差距。

总之，在沟通活动中收集到的信息作为需求模型的输入，即从非正式的电子邮件到包括综合使用场景和产品规格说明书中详细项目概述的任何内容。

11.5.3 需求建模的输出

需求分析提供了严格规定的机制来描述和评估 App 的内容和功能、用户将遇到的交互模式，以及 Web/移动 App 所处的环境和基础设施。

每种特性都能表示成一套模型，这套模型允许以结构化方式分析 App 的需求。当特定模型很大程度上依赖于 App 的性质时，会有 5 种主要的模型类型：

- **内容模型**给出由 App 提供的全部内容，包括文本、图表、图像、视频和音频数据。

① 如果 App 中的一部分设计会对其他部分产生影响，那么分析的范围就应当扩大。

- 交互模型描述了用户与 App 采用了哪种交互方式。
- 功能模型定义了将用于处理 App 内容并描述其他处理功能的操作，这些处理功能不依赖于内容，但却是最终用户所必需的。
- 导航模型为 App 定义所有导航策略。
- 配置模型描述 App 所在的环境和基础设施。

分析师可以使用描述模式开发每一种模型，这种常称为“语言”的描述模式考虑到其意图和结构，使工程团队的成员和相关利益者之间更容易进行沟通和评估。最终识别出关键问题列表（例如错误、遗漏、不一致性、供增强和更改的建议、澄清观点）并照此行动。

11.5.4 内容模型

内容模型包含结构元素，它们为 App 的内容需求提供了一个重要的视图。这些结构元素包含内容对象和所有分析类，在用户通过浏览器或移动设备与 App 交互时生成并操作用户可见的实体^①。

内容的开发可能发生在 App 实现之前、App 构建之中，或者 App 投入运行以后的很长一段时间里。在每种情况下，它都通过导航链接合并到 App 的总体结构中。内容对象可能是产品的文本描述、描述新闻事件的一篇文章、抽取数据的图形表示（例如随时间波动的股票价格）、在体育运动事件中拍摄的一张动作照片、在一次论坛讨论中某个用户的回答、公司徽标的一种生动演示、一个演讲的简短视频，或者是一套演示幻灯片中的音频插曲。这些对象内容可能存储于分离的文件中，或从数据库中动态获得，也可能直接嵌入 Web 页中，显示在移动设备屏幕上。换句话说，内容对象是呈现给最终用户的具有内聚信息的所有条目。

通过检查直接或间接引用内容的场景描述，可以根据用例直接确定出内容对象。例如，建立在 www.safehomeassured.com 中支持 SafeHome 的 WebApp。用例选择 SafeHome 构件描述了需要购买 SafeHome 构件的场景并包含如下句子：

我将能得到每种产品构件的说明和价格信息。

内容模型必须具备描述内容对象构件的能力。在许多实例中，用一个简单的内容对象列表，并给出每个对象的简短描述就足以定义设计和实现所必需的内容需求。但是在某些情况下，内容模型可以从更丰富的分析中获得好处，即在内容对象之间和 WebApp 维护的内容层次中采用图形化方法表示其关系。

例如，考虑图 11-5 中为 www.safehomeassured.com 构件建立的数据树 [Sri01]。该树表述了常用于描述某个构件的一种信息层次。不带阴影的长方形表示简单或复合数据项（一个或多个数据值），带阴影的长方形表示内容对象。在此图中，描述说明是由 5 个内容对象（有阴影的长方形）定义的。在某些情况下，随着数据树的扩展，其中的一个或多个对象将会得到进一步细化。

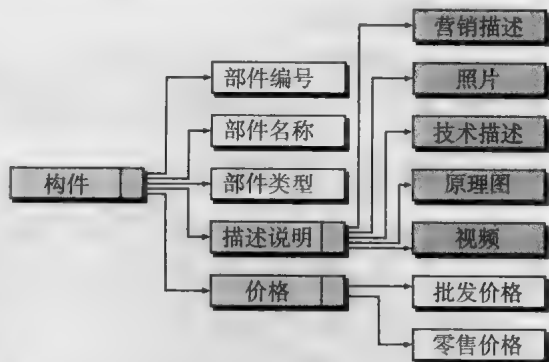


图 11-5 www.safehomeassured.com 构件的数据树

① 分析类已在第 10 章讨论。

由多项内容对象和数据项组成的任何内容都可以生成数据树。开发数据树时要尽力在内容对象中定义层级关系，并提供一种审核内容的方法，从而在开始设计前发现遗漏的和不一致的内容。另外，数据树可以作为内容设计的基础。

11.5.5 WebApp 和移动 App 的交互模型

绝大多数 Web 和移动 App 都能使最终用户与 App 的功能、内容及行为之间进行“会话”。可以使用交互模型描述会话，这种交互模型由一种或多种元素组成：（1）用例；（2）顺序图；（3）状态图[⊖]；（4）用户界面原型。

在许多实例中，一套用例[⊕]足以描述在分析阶段的所有交互活动（在设计阶段引入更精确的细节）。然而当遇到复杂的交互顺序并包含多个分析类或多任务时，有时更值得采用严格图解方式描述它们。

[217]

用户界面的布局、介绍的内容、实现的交互机制以及用户与 WebApp 连接上的总体审美观，都与用户的满意度和 App 的整体成功密切相关。虽然有理由认为用户界面原型的创建是一种设计活动，但在创建分析模型期间执行用户界面原型的创建仍是一个好办法。对用户界面的物理表示评估得越早，就越有可能满足最终用户的需求。在第 15 章将详细讨论用户界面的设计。

因为 Web 和移动 App 的构造工具种类繁多，而且相对便宜、功能强大，因此最好使用这些工具创建界面原型。原型应该实现主要的导航链接，并采用同样的构造方式表现总体的屏幕布局。例如，为用户提供 5 种主要系统功能，当用户第一次进入 App 时，这些原型就应呈现出这些功能。如果要问将会提供图形链接吗？导航菜单会显示在哪里？用户会看到哪些其他信息？可以由原型回答以上类似的问题。

11.5.6 功能模型

许多 WebApp 都提供大量的计算和操作功能，这些功能与内容直接相关（既能使用又能生成内容）。这些功能常常以用户 - WebApp 的交互活动为主要目标。移动 App 更趋于关注和提供较有限的计算能力和操作功能的集合。但无论功能广度如何，都必须分析功能需求，并且在需要时也可以用于建模过程。

功能模型描述 WebApp 的两个处理元素，每个处理元素代表抽象过程的不同层次：（1）用户可观察到的功能是由 WebApp 传递给最终用户的；（2）分析类中的操作实现与类相关的行为。

用户可观察到的功能包括直接由用户启动的任何处理功能。例如，金融 WebApp 可以执行多种财务功能（例如支付抵押品的计算）。这些功能实际上可使用分析类中的操作完成，但是从最终用户的角度看，这些功能（更正确地说，这些功能所提供的数据）是可见的结果。

在抽象过程的更低层次，分析模型描述了由分析类操作执行的处理。这些操作可以操纵类的属性，并参与类之间的协作来完成所需要的行为。

不管抽象过程的层次如何，UML 的活动图都可用来表示处理细节。在分析阶段，仅在功能相对复杂的地方才会使用活动图。许多 WebApp 的复杂性不是呈现在提供的功能中，而是与可访问信息的性质以及操作的方式相关。

[218]

⊖ 使用 UML 符号为顺序图或状态图建模。

⊕ 用例的详细描述见第 9 章。

例如，在 www.safehomeassured.com 中，一个相对复杂的功能可以由一个用例来描述，这个用例名为为我的空间中传感器的布局提供建议。用户已经开发了一个可以监控空间的布局，在该用例中选择该布局，并要求传感器置于建议的位置。www.safehomeassured.com 用该布局的图形化表示方法回应，为推荐位置的传感器提供额外的信息。交互活动是非常简单的，某些内容有些复杂，然而它的底层功能非常复杂。系统必须承担地面布局的复杂分析，以便决定传感器的优化位置。必须检查房间面积、门窗位置，并协调这些传感器的功能和技术要求。这绝不是个小任务！我们使用一套活动图来描述这个用例所做的处理。

第二个例子是名为控制摄像机的用例。在这个用例中，交互活动相对简单，但存在潜在的复杂功能，这些“简单”操作需要通过访问互联网和远程设备进行复杂通信。当有多个已经授权的人同时想监视并且控制某个传感器时，便可能出现更复杂的情况，这与控制权协商有关。

图 11-6 为 `takeControlOfCamera()` 操作描绘了一幅活动图，它是控制摄像机用例中所用摄像机分析类的一部分。应该注意到程序流程中包括两项额外操作：`requestCameraLock()` 想为这个用户锁定摄像机；`getCurrentCameraUser()` 获取当前控制这台摄像机的用户名字。暂不考虑描述如何调用这些操作的结构细节和每项操作的接口细节，这些直到 WebApp 设计阶段才会给出建议。

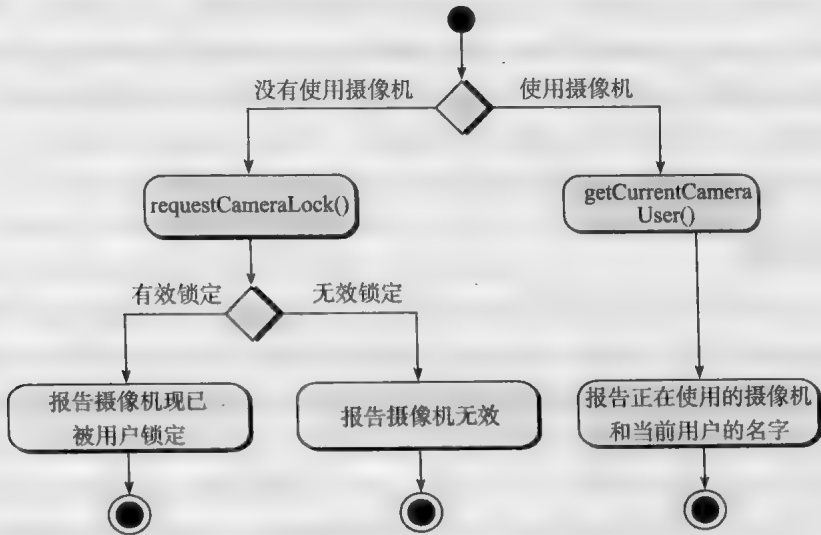


图 11-6 `takeControlOfCamera()` 操作的活动图

SafeHome WebApp 的扩展功能可能会涉及移动 App 的开发，提供了从智能手机或者平板电脑进入 SafeHome 系统的功能。SafeHome 移动 App 的内容和功能需求可与 WebApp 的子集相似，但还必须建立特殊接口和安全需求。

11.5.7 WebApp 的配置模型

219

在某些情况下，配置模型只不过是服务器端和客户端的属性列表。但是，对更复杂的 App 来说，多种配置的复杂性（例如，多服务器之间的负载分配、高速缓存的体系结构、远程数据库、服务于不同对象的多个服务器）可能对分析和设计产生影响。在必须考虑复杂配置体系结构的情况下，可以使用 UML 部署图。

应该指定通过所有主要的 Web 客户端访问 www.safehomeassured.com 的公共内容和功

能（例如，那些多于 1% 市场占有率的 Web 客户端）。相反，对于少量客户端也可限制复杂的控制和监控功能（只允许 HomeOwner 用户访问）。对移动 App 而言，实施方案可能局限于三种领先的移动操作环境中。www.safehomeassured.com 的配置模型也将与指定的现存产品数据库和监控应用进行交互操作。

11.5.8 导航建模

对于智能手机平台上的大多数移动 App，导航通常局限于相对简单的按键列表和图标菜单。另外，导航的深度（例如超链接的层级数）相对较浅。由于这些原因，导航模型相对简单。

随着 WebApp 和基于平板的移动 App 数量的不断增长，导航建模将更为复杂，常常需要考虑如何让每个用户分类从一个 WebApp 元素（例如内容对象）跳转到另一个元素。我们把导航机制定义为设计的一部分。在这个阶段，分析人员应该关注总体的导航需求，考虑以下问题：

220

- 某些元素比其他元素更容易到达吗（即需要更少的导航步骤）？表示的优先级别是什么？
- 为了促使用户按他们自己的方向导航，应该强调某些元素吗？
- 应该怎样处理导航错误？
- 导航到相关元素组的优先级应该高于导航到某个特定元素的优先级吗？
- 应该通过链接方式、基于搜索的访问方式还是其他方式来实现导航？
- 根据前面的导航行为，某些确定的元素应该展现给用户吗？
- 应该为用户维护导航日志吗？
- 在用户交互的每一点处，（与单个“返回”链接或有方向的指针相比）完整的导航地图或菜单都可用吗？
- 导航设计应该由大多数普遍期望的用户行为来驱动，还是由已定义的 WebApp 元素可感知的重要性来驱动？
- 为了使将来的使用更加快捷，用户能否“存储”他以前对 WebApp 的导航？
- 应该为哪类用户设计最佳导航？
- 应该如何处理 WebApp 外部的链接？应该覆盖现有的浏览器窗口吗？能否作为一个新的浏览器窗口，还是作为一个单独的框架？

提出并回答这些问题及其他一些问题是导航分析工作的一部分。

软件工程师和其他利益相关者必须决定导航的总体需求。例如，“站点图”会让用户全面纵览整个 WebApp 的结构吗？用户会使用“导游”吗？“导游”将突出显示可以使用的最重要的元素（包括内容对象和功能）吗？用户能够访问内容对象或者由其元素属性所决定的功能吗（例如，用户可能想要访问一个特定建筑的所有图片，或者允许计算重量的所有功能）？

11.6 小结

在需求分析阶段的行为建模描述了软件的动态行为。行为模型采用来自于基于场景、面向流和基于类的输入，从而把分析类和系统的状态作为一个整体来表达。为达到这一目的，要识别状态，定义引起类（或系统）由一个状态转换到另一状态的事件，还要识别完成转换后发生的活动。状态图和顺序图是行为建模的常用表达方式。

221

分析模式使得软件工程师能够使用现有的知识领域，便于建立需求模型。一个分析模式

描述了一个特定的软件特性或功能，该特性或功能由一套相关用例描述。本章详细说明了模式的目的、使用的动机、使用的限制约束、各种问题域中的可应用性、模式的完整结构、行为和协作以及其他辅助信息。

WebApp 的需求建模能使用本书讨论的大多数的建模元素。但是只能在一套指定的模型中使用这些元素，它涉及 WebApp 中的内容、交互操作、功能、导航和服务客户端的配置。

习题与思考题

- 11.1 表示行为建模时有两种不同“状态”类型，它们是什么？
- 11.2 如何从状态图区分顺序图？它们有何相似之处？
- 11.3 为现代移动手机建议 3 种需求模式，并简要描述每种模式。这些模式能否被其他设备使用？举例说明。
- 11.4 在问题 11.3 中选择一个你要开发的模式，模拟 11.4.2 节中表达的某个内容和模型，开发出一个合理并完整的模式描述。
- 11.5 你认为 www.safehomeassured.com 需要多少种分析模型？这些是在 11.5.3 节中描述的每种模型类型所需要的吗？
- 11.6 WebApp 交互建模的目的是什么？
- 11.7 引起争议的问题是 WebApp 的功能模型应延期开发直至设计阶段。表述关于这个议题的支持和反对观点。
- 11.8 配置模型的目的是什么？
- 11.9 如何从交互模型中区分导航模型？

扩展阅读与信息资源

行为建模呈现了系统行为的动态视图。Samek (《Practical UML Statecharts in C/C++: Event Driven Programming for Embedded Systems》，CRC Press, 2008)、Wagner 和他的同事 (《Modeling Software with Finite State Machines: A Practical Approach》，Auerbach, 2006) 以及 Boerger 和 Staerk (《Abstract State Machines》，Springer, 2003) 深入讨论了状态图和其他行为的表示方法。Gomes 和 Fernandez (《Behavioral Modeling for Embedded Systems and Technologies》，Information Science Reference, 2009) 为描述嵌入式系统的行为模型编辑了一本选集。

222

为软件模式所写的大部分书都关注软件设计。而 Vaughn (《Implementing Domain-Driven Design》，Addison-Wesley, 2013)、Whithall (《Software Requirement Patterns》，Microsoft Press, 2007)、Evans (《Domain-Driven Design》，Addison-Wesley, 2003) 和 Fowler ([Fow03], [Fow97]) 的书都是专门写分析模式的。

Pressman 和 Lowe[Pre08] 表述了需要深层次讨论的 WebApp 分析模型。Rossi 和他的同事 (《Web Engineering: Modeling and Implementing Web Applications》，Springer, 2010) 以及 Neil (《Mobile Design Pattern Gallery: UI Patterns》，O'Reilly, 2012) 讨论在应用开发中模式的使用方法。Murugesan 和 Desphande 编写的论文集中包含了一篇文章 (《Web Engineering: Managing Diversity and Complexity of Web Application Development》，Springer, 2001)，处理了 WebApp 需求的各个方面。另外《International Conference on Web Engineering》论文集每年都会发表解决需求建模问题的文章。

网上有大量需求建模方面的资源。关于分析建模的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 上找到。

223

设计概念

要点浏览

概念：几乎每位工程师都希望做设计工作。设计体现了创造性——利益相关者的需求、业务要求和技术考虑都集中地体现在产品或系统的形成中。设计创建了软件的表示或模型，但与需求模型（注重描述所需的数据、功能和行为）不同，设计模型提供了软件体系结构、数据结构、接口和构件的细节，而这些都是实现系统所必需的。

人员：软件工程师负责处理每一项设计任务。

重要性：设计是让软件工程师为将要构建的系统或产品建立模型。在生成代码、进行测试以及大量最终用户使用之前，要对模型的质量进行评估，并进行改进。软件质量是在设计中建立的。

步骤：设计可以采用很多不同的方式描述

软件。首先，必须表示系统或产品的体系结构；其次，为各类接口建模，这些接口在软件和最终用户、软件和其他系统与设备以及软件和自身组成的构件之间起到连接作用；最后，设计构成系统的软件构件。每个视图表现了不同的设计活动，但所有的视图都要遵循一组指导软件设计工作的基本设计概念。

工作产品：在软件设计过程中，包含体系结构、接口、构件级和部署表示的设计模型是主要的工作产品。

质量保证措施：软件团队从以下各方面来评估设计模型：确定设计模型是否存在错误、不一致或遗漏；是否存在更好的可选方案；设计模型是否可以在已经设定的约束、时间进度和成本内实现。

软件设计包括一系列原理、概念和实践，可以指导高质量的系统或产品开发。设计原理建立了指导设计工作的最重要原则。在运用设计实践的技术和方法之前，必须先理解设计概念，设计实践本身会产生软件的各种表示，以指导随后的构建活动。

设计是软件工程是否成功的关键。在 20 世纪 90 年代早期，Lotus 1-2-3 的创始人 Mitch Kapor 在《Dr. Dobbs》杂志上发表了如下“软件设计宣言”：

什么是设计？设计是你身处两个世界——技术世界和人类的目标世界，而你尝试将这两个世界结合在一起……

罗马建筑批评家 Vitruvius 提出了这样一个观念：设计良好的建筑应该展示出坚固、适用和令人愉悦的特点。对好的软件来说也是如此。坚固：程序应该不含任何妨碍其功能的缺陷。适用：程序应该符合开发的目标。愉悦：使用程序的体验应是愉快的。本章，我们开始介绍软件设计理论。

关键概念

抽象
体系结构
方面
内聚
数据设计
设计过程
功能独立
良好的设计
信息隐蔽
模块化
面向对象的设计
模式
质量属性
质量指导原则

设计的目标是创作出坚固、适用和令人愉悦的模型或表示。为此，设计师必须先实现多样化，然后再进行聚合。Belady[Bel81]指出，“多样化是指要获取所有方案和设计的原始资料，包括目录、教科书和头脑中的构件、构件方案和知识。”在各种信息汇聚在一起之后，应从满足需求工程和分析模型（第8～11章）所定义的需求中挑选适合的元素。此时，考虑并取舍候选方案，然后进行聚合，使之成为“构件的某种特定的配置，从而创建最终的产品”[Bel81]。

多样化和聚合需要直觉和判断力，其质量取决于构建类似实体的经验、一系列指导模型演化方式的原则和启发、一系列质量评价的标准以及导出最终设计表示的迭代过程。

新的方法不断出现，分析过程逐步优化，人们对设计的理解也日渐广泛，随之而来的是软件设计的发展和变更^①。即使是今天，大多数软件设计方法都缺少那些更经典的工程设计学科所具有的深度、灵活性和定量性。然而，软件设计的方法是存在的，设计质量的标准是可以获得的，设计表示法也是能够应用的。在本章中，我们将探讨可以应用于所有软件设计的基本概念和原则、设计模型的元素以及模式对设计过程的影响。在第12～18章中，我们将介绍应用于体系结构、接口和构件级设计的多种软件设计方法，也会介绍基于模式和面向Web的设计方法。

关键概念

重构
关注点分离
软件设计
逐步求精

引述 软件工程

最常见的奇迹是从分析到设计以及从设计到代码的转换。

Richard Due'

12.1 软件工程中的设计

软件设计在软件工程过程中属于核心技术，并且它的应用与所使用的软件过程模型无关。一旦对软件需求进行分析和建模，软件设计就开始了。软件设计是建模活动的最后一个软件工程活动，接着便要进入构建阶段（编码和测试）。

需求模型的每个元素（第9～11章）都提供了创建四种设计模型所必需的信息，这四种设计模型是完整的设计规格说明所必需的。软件设计过程中的信息流如图12-1所示。由基于场景的元素、基于类的元素和行为元素所表示的需求模型是设计任务的输入。使用后续章节所讨论的设计表示法和设计方法，将得到数据或类的设计、体系结构设计、接口设计和构件级设计。

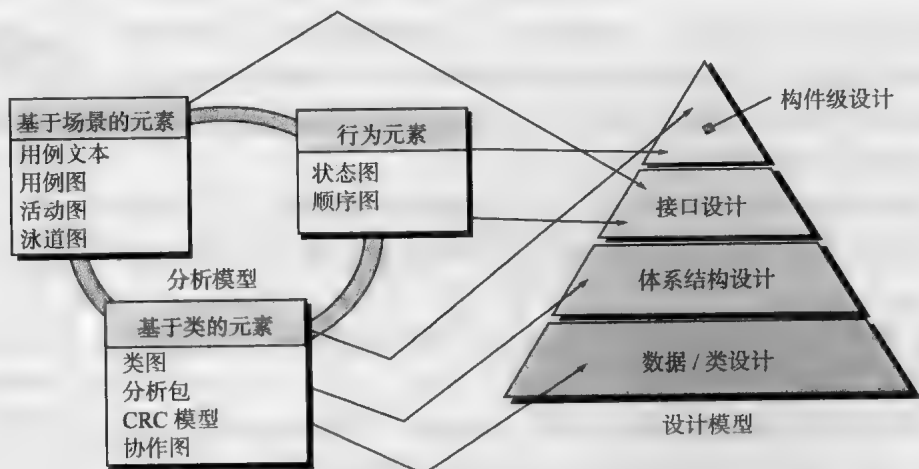


图 12-1 从需求模型到设计模型的转换

① 对软件设计原理感兴趣的读者可能会对 Philippe Kruchten 关于“后现代”设计的有趣讨论感兴趣 [Kru05]。

数据设计或类设计将类模型(第10章)转化为设计类的实现以及软件实现所要求的数据结构。CRC图中定义的对象和关系,以及类属性和其他表示法描述的详细数据内容为数据设计活动提供了基础。在软件体系结构设计中也可能会进行部分类的设计,更详细的类设计则将在设计每个软件构件时进行。

体系结构设计定义了软件的主要结构化元素之间的关系、可满足系统需求的体系结构风格和模式(第13章)以及影响体系结构实现方式的约束[Sha96]。体系结构设计表示可以从需求模型导出,该设计表示基于的是计算机系统的框架。

接口设计描述了软件和合作系统之间、软件和使用人员之间是如何通信的。接口意味着信息流(如数据和控制)和特定的行为类型。因此,使用场景和行为模型为接口设计提供了大量的信息。

构件级设计将软件体系结构的结构化元素变换为对软件构件的过程性描述。从基于类的模型和行为模型中获得的信息是构件设计的基础。

设计过程中所做出的决策将最终影响软件构建的成功与否,更重要的是,会影响软件维护的难易程度。但是,设计为什么如此重要呢?

软件设计的重要性可以用一个词来表达——质量。在软件工程中,设计是质量形成的地方,设计提供了可以用于质量评估的软件表示,设计是将利益相关者的需求准确地转化为最终软件产品或系统的唯一方法。软件设计是所有软件工程活动和随后的软件支持活动的基础。没有设计,将会存在构建不稳定系统的风险,这样的系统稍做改动就无法运行,而且难以测试,直到软件过程的后期才能评估其质量,而那时时间已经不够并且已经花费了大量经费。

建议 软件设计应该总是先考虑数据,数据是所有其他设计元素的基础。基础奠定之后,才能进行体系结构设计。再之后,才能进行其他设计任务。

引述 有两种构建软件设计的方式:一种是使其尽可能简单以致明显没有不足;另一种是使其尽可能复杂以致没有明显的不足。第一种方式更为困难。

C. A. R Hoare

226

SafeHome 设计与编码

[场景] Jamie 的房间,团队成员准备将需求转化为设计。

[人物] Jamie、Vinod 和 Ed, SafeHome 软件工程团队所有成员。

[对话]

Jamie: 大家都知道, Doug (团队管理者)沉迷于设计。老实说,我真正喜欢的是编码。如果给我 C++ 或者 Java,我会非常高兴。

Ed: 不,你喜欢设计。

Jamie: 你没听我说吗?编码才是我喜欢的。

Vinod: 我想 Ed 的意思是你并不是真的喜欢编码,而是喜欢设计,并喜欢用代码表达设计。代码是你用来表示设计的

语言。

Jamie: 那有什么问题吗?

Vinod: 抽象层。

Jamie: 嗯?

Ed: 程序设计语言有利于表示诸如数据结构和算法的细节,但不利于表示体系结构或者构件之间的协作……就是这个意思。

Vinod: 一个糟糕的体系结构甚至能够摧毁最好的代码。

Jamie(思考片刻): 那么,你们的意思是我不能用代码表示体系结构……这不是事实。

Vinod: 你肯定能在代码中隐含体系结构,但在大部分程序设计语言中,通过检查

代码而快速看到体系结构的全貌是相当困难的。

Jamie：我同意，也许设计和编码不同，但我仍然更喜欢编码。

Ed：那正是我们在开始编码之前需要的。

227

12.2 设计过程

软件设计是一个迭代的过程，通过这个过程，需求被变换为用于构建软件的“蓝图”。刚开始，蓝图描述了软件的整体视图，也就是说，设计是在高抽象层次上的表达——在该层次上可以直接跟踪到特定的系统目标以及更详细的数据、功能和行为需求。随着设计迭代的开始，后续的细化导致更低抽象层次的设计表示。这些表示仍然能够跟踪到需求，但是连接更加错综复杂了。

12.2.1 软件质量指导原则和属性

在整个设计过程中，我们使用第 20 章中讨论的一系列技术评审来评估设计演化的质量。McGlaughlin[McG91] 提出了可以指导良好设计演化的三个特征：

- 设计应当实现所有包含在需求模型中的明确需求，而且必须满足利益相关者期望的所有隐含需求。
- 对于那些编码者和测试者以及随后的软件维护者而言，设计应当是可读的、可理解的指南。
- 设计应当提供软件的全貌，从实现的角度对数据域、功能域和行为域进行说明。

以上每一个特征实际上都是设计过程的目标，但是如何达到这些目标呢？

质量指导原则。为了评估某个设计表示的质量，软件团队中的成员必须建立良好设计的技术标准。在 12.3 节中，我们将讨论设计概念，这些概念也可以作为软件质量的标准。现在，考虑下面的指导原则：

1. 设计应展现出这样一种体系结构：（1）已经使用可识别的体系结构风格或模式创建；（2）由能够展现出良好设计特征的构件构成（将在本章后面讨论）；（3）能够以演化的方式^①实现，从而便于实施与测试。
2. 设计应该模块化，也就是说，应将软件逻辑地划分为元素或子系统。
3. 设计应该包含数据、体系结构、接口和构件的清晰表示。
4. 设计应导出数据结构，这些数据结构适用于要实现的类，并从可识别的数据模式提取。
5. 设计应导出显示独立功能特征的构件。
6. 设计应导出接口，这些接口降低了构件之间以及构件与外部环境之间连接的复杂性。
7. 设计的导出应采用可重复的方法进行，这些方法由软件需求分析过程中获取的信息而产生。

228

引述 编写一段能工作的灵巧的代码是一回事，而设计能支持某个长久业务的东西则完全是另一回事。

C. Ferguson

提问 良好设计的特征是什么？

引述 设计不仅仅是它看上去以及感觉上如何，还要看它是如何运作的。

Steve Jobs

① 对于较小的系统，设计有时也可以线性地进行开发。

8. 应使用能够有效传达其意义的表示法来表达设计。

满足这些设计原则并非依靠偶然性，而是通过应用基本的设计原则、系统化的方法学和严格的评审来得到保证。

信息栏 评估设计质量——技术评审

设计之所以重要，在于它允许一个软件团队在软件实现前评估软件的质量^①——此时修正错误、遗漏或不一致都不困难且代价不高。但是我们如何在设计过程中评估质量呢？此时，不可能去测试软件，因为还没有可执行的软件，那怎么办？

在设计阶段，可以通过开展一系列的技术评审（Technical Review，TR）来评估质量。技术评审将在第20章^②中进行详细讨论，这里先概述一下该技术。技术评审是由软件团队成员召开的会议。通常，根据将要评审的设计信息的范围，选择2~4人参与。每人扮演一个角色：

评审组长策划会议、拟定议程并主持会议；记录员做笔记以保证没有遗漏；制作者是指其工作产品（例如某个软件构件的设计）被评审的人。在会议之前，评审小组的每个成员都会收到一份设计工作产品的拷贝并要求阅读，寻找错误、遗漏或含糊不清的地方。目的是在会议开始时注意到工作产品中的所有问题，以便能够在开始实现该产品之前修正这些问题。技术评审通常持续60~90分钟。评审结束时，评审小组要确认在设计工作产品能够被认可为最终设计模型的一部分之前，是否还需要进一步的行动。

229

质量属性。Hewlett-Packard[Gra87]制订了一系列的软件质量属性，并取其首字母组合为FURPS，其中各字母分别代表功能性（functionality）、易用性（usability）、可靠性（reliability）、性能（performance）及可支持性（supportability）。FURPS质量属性体现了所有软件设计的目标：

- 功能性通过评估程序的特征集和能力、所提交功能的通用性以及整个系统的安全性来评估。
- 易用性通过考虑人员因素（第6~15章）、整体美感、一致性和文档来评估。
- 可靠性通过测量故障的频率和严重性、输出结果的精确性、平均故障时间（Mean-Time-To-Failure，MTTF）、故障恢复能力和程序的可预见性来评估。
- 性能通过考虑处理速度、响应时间、资源消耗、吞吐量和效率来度量。
- 可支持性综合了可扩展性、可适应性和可用性。这三个属性体现了一个更通用的术语：可维护性。此外，还包括可测试性、兼容性、可配置性（组织和控制软件配置元素的能力，第29章）、系统安装的简易性和问题定位的容易性。

引述 质量不是那些被束之高阁的观赏品，也不是像圣诞树上的闪亮金箔那样的装饰品。

Robert Pirsig

建议 软件设计师往往注重于问题的解决。不要忘记的是：FURPS属性总是问题的一部分，因此必须要考虑到这些属性。

① 第30章讨论的质量因素可以帮助评审小组评估质量。

② 设计时，可以提前看一下第20章。技术评审是设计过程中的关键部分，也是达到设计质量的重要方法。

在进行软件设计时，并不是每个软件质量属性都具有相同的权重。有的应用问题可能强调功能性，特别突出安全性；有的应用问题可能要求性能，特别突出处理速度；还有的可能关注可靠性。抛开权重不谈，重要的是：必须在设计开始时就考虑这些质量属性，而不是在设计完成后和构建已经开始时才考虑。

12.2.2 软件设计的演化

软件设计的演化是一个持续的过程，它已经经历了 60 多年的发展。早期的设计工作注重模块化程序开发的标准 [Den73] 和以自顶向下的“结构化”方式对软件结构进行求精的方法 ([Wir71], [Dah72], [Mil72])。较新的设计方法 (如 [Jac92]、[Gam95]) 提出了进行设计导出的面向对象方法。近年来，软件体系结构 [Kru06] 和可用于实施软件体系结构及较低级别设计抽象 (如 [Hol06]、[Sha05]) 的设计模式已经成为软件设计的重点。面向方面的方法 (如 [Cla95]、[Jac04])、模型驱动开发 [Sch06] 以及测试驱动开发 [Ast04] 日益受到重视，这些方法强调在设计中实现更有效的模块化和体系结构的技术。

引述 设计师知道当设计中不再需要减去任何东西，而并不是不再需要增加任何东西的时候，他的设计就很完美了。

Antoine de
St-Exupery

提问 所有设计方法的共同设计特征是什么？

许多设计方法刚刚被提出，它们从工作中产生，并正被运用于整个行业。正如第 9 ~ 11 章提出的分析方法，每一种软件设计方法引入了独特的启发式和表示法，同时也引入了某种标定软件质量特征的狭隘观点。不过，这些方法都有一些共同的特征：(1) 将需求模型转化为设计表示的方法；(2) 表示功能性构件及它们之间接口的表示法；(3) 细化和分割的启发式方法；(4) 质量评估的指导原则。

无论使用哪种设计方法，都应该将一套基本概念运用到数据设计、体系结构设计、接口设计和构件级设计，这些基本概念将在后面几节中介绍。

任务集 通用设计任务集

1. 检查信息域模型，并为数据对象及其属性设计合适的数据结构。
2. 使用分析模型选择一种适用于软件的体系结构风格 (模式)。
3. 将分析模型分割为若干设计子系统，并在体系结构内分配这些子系统：
 - 确保每个子系统是功能内聚的。
 - 设计子系统接口。
 - 为每个子系统分配分析类或功能。
4. 创建一系列的设计类或构件：
 - 将分析类描述转化为设计类。
 - 根据设计标准检查每个设计类，考虑继承问题。
 - 定义与每个设计类相关的方法和消息。
- 评估设计类或子系统，并为这些类或子系统选择设计模式。
- 评审设计类，并在需要时进行修改。
5. 设计外部系统或设备所需要的所有接口。
6. 设计用户接口：
 - 评审任务分析的结果。
 - 基于用户场景对活动序列进行详细说明。
 - 创建接口的行为模型。
 - 定义接口对象和控制机制。
 - 评审接口设计，并根据需要进行修改。
7. 进行构件级设计：
 - 在相对较低的抽象层次上详细描述

- 所有算法。
- 细化每个构件的接口。
- 定义构件级的数据结构。
- 评审每个构件并修正所有已发现的错误。
- 8. 开发部署模型。

12.3 设计概念

在软件工程的历史上，产生了一系列基本的软件设计概念。尽管多年来人们对于这些概念的关注程度不断变化，但它们都经历了时间的考验。每一种概念都为软件设计者应用更加复杂的设计方法提供了基础。每种方法都有助于：定义一套将软件分割为独立构件的标准，从软件的概念表示中分离出数据结构的细节，为定义软件设计的技术质量建立统一标准。

231

M. A. Jackson[Jac75] 曾经说过：“软件工程师的智慧开始于认识到‘使程序工作’和‘使程序正确’之间的差异。”在后面几节中，将对基本的软件设计概念进行介绍，这些概念为“使程序正确”提供了必要的框架。

12.3.1 抽象

当考虑某一问题的模块化解决方案时，可以给出许多抽象级。在最高的抽象级上，使用问题所处环境的语言以概括性的术语描述解决方案。在较低的抽象级上，将提供更详细的解决方案说明。当力图陈述一种解决方案时，面向问题的术语和面向实现的术语会同时使用。最后，在最低的抽象级上，以一种能直接实现的方式陈述解决方案。

引述 抽象是人类处理复杂问题的基本方法之一。

Grady Booch

在开发不同层次的抽象时，软件设计师力图创建过程抽象和数据抽象。过程抽象是指具有明确和有限功能的指令序列。“过程抽象”这一命名暗示了这些功能，但隐藏了具体的细节。过程抽象的例子如开门。开隐含了一长串的过程性步骤（例如，走到门前，伸出手并抓住把手，转动把手并拉门，从移动门走开等）。^①

建议 作为设计师，致力于得到解决现有问题的过程抽象和数据抽象，但如果他们能在整个问题域中起作用，那就更好了。

数据抽象是描述数据对象的具名数据集合。在过程抽象开的场景下，我们可以定义一个名为 door 的数据抽象。同任何数据对象一样，door 的数据抽象将包含一组描述门的属性（例如，门的类型、转动方向、开门方法、重量和尺寸）。因此，过程抽象开将利用数据抽象 door 的属性中所包含的信息。

12.3.2 体系结构

软件体系结构意指“软件的整体结构和这种结构为系统提供概念完整性的方式”[Sha95a]。从最简单的形式来看，体系结构是程序构件（模块）的结构或组织、这些构件交互的方式以及这些构件所用数据的结构。然而在更广泛的意义上，构件可以概括为主要的系统元素及其交互方式的表示。

网络资源 关于软件体系结构深入的讨论可以在 www.sei.cmu.edu/ata/ata_init.html 找到。

232

软件设计的目标之一是导出系统体系结构示意图，该示意图作为一个

① 然而，需要注意的是：只要过程抽象隐含的功能相同，一组操作就可以被另一组操作代替。因此，如果门是自动的并连接到传感器上，那么实现“开”所需的步骤将会完全不同。

框架,将指导更详细的设计活动。一系列的体系结构模式使软件工程师能够重用设计层概念。

Shaw 和 Garlan[Sha95a]描述了一组属性,这组属性应该作为体系结构设计的一部分进行描述。结构特性定义了“系统的构件(如模块、对象、过滤器)、构件被封装的方式以及构件之间相互作用的方式”。外部功能特性指出“设计体系结构如何满足需求,这些需求包括性能需求、能力需求、可靠性需求、安全性需求、可适应性需求以及其他系统特征需求”。相关系统族“抽取相似系统设计中常遇到的重复性模式”。

一旦给出了这些特性的规格说明,就可以用一种或多种不同的模型来表示体系结构设计[Gar95]。结构模型将体系结构表示为程序构件的有组织的集合。框架模型可以通过确定相似应用中遇到的可复用体系结构设计框架(模式)来提高设计抽象的级别。动态模型强调程序体系结构的行为方面,指明结构或系统配置如何随着外部事件的变化而产生变化。过程模型强调系统必须提供的业务或技术流程的设计。最后,功能模型可用于表示系统的功能层次结构。

为了表示以上描述的模型,人们已经开发了许多不同的体系结构描述语言(Architectural Description Language, ADL)[Sha95b]。尽管提出了许多不同的ADL,但大多数ADL都提供描述系统构件和构件之间相互联系方式的机制。

需要注意的是,关于体系结构在设计中的地位还存在一些争议。一些研究者主张将软件体系结构的设计从设计中分离出来,并在需求工程活动和更传统的设计活动之间进行。另外一些研究者认为体系结构设计是设计过程不可分割的一部分。第13章将讨论软件体系结构特征描述方式及软件体系结构在设计中的作用。

12.3.3 模式

Brad Appleton 以如下方式定义设计模式:“模式是具名的洞察力财宝,对于竞争事件中某确定环境下重复出现的问题,它承载了已证实的解决方案的精髓”[App00]。换句话说,设计模式描述了解决某个特定设计问题的设计结构,该设计问题处在一个特定环境中,该环境会影响到模式的应用和使用方式。

每种设计模式的目的是提供一种描述,以使设计人员可以决定:(1)模式是否适用于当前的工作;(2)模式是否能够复用(因此节约设计时间);(3)模式是否能够用于指导开发一个相似的但功能或结构不同的模式。设计模式将在第16章进行详细讨论。

12.3.4 关注点分离

关注点分离是一个设计概念[Dij82],它表明任何复杂问题如果被分解为可以独立解决或优化的若干块,该复杂问题便能够更容易地得到处理。关注点是一个特征或一个行为,被指定为软件需求模型的一部分。将关注点分割为更小的关注点(由此产生更多可管理的块),便可用更少的工作量和时间解决一个问题。

另一个结果是:两个问题被结合到一起的认知复杂度经常高于每个问题各自的认知复杂度之和。这就引出了“分而治之”的策略——把一个复杂问题分解为若干可管理的块来求解

引述 软件体系结构是在质量、进度和成本方面具有最高投资回报的工作产品。

Len Bass et al.

引述 每个模式都描述了一个在我们所处环境内反复发生的问题,然后描述该问题的核心解决方案,你可以几百万次地重复使用该解决方案,而根本不需要用同样的方式重复工作两次。

Christopher Alexander

时将会更容易。这对于软件的模块化具有重要的意义。

关注点分离在其他相关设计概念中也有体现：模块化、方面、功能独立、求精。每个概念都会在后面的小节中讨论。

12.3.5 模块化

模块化是关注点分离最常见的表现。软件被划分为独立命名的、可处理的构件，有时被称为模块，把这些构件集成到一起可以满足问题的需求。

有人提出“模块化是软件的单一属性，它使程序能被智能化地管理”[Mye78]。软件工程师难以掌握单块软件（即由一个单独模块构成的大程序）。对于单块大型程序，其控制路径的数量、引用的跨度、变量的数量和整体的复杂度使得理解这样的软件几乎是不可能的。绝大多数情况下，为了理解更容易，都应当将设计划分成许多模块，这样做的结果是降低构建软件所需的成本。

回顾关于关注点分离的讨论，可以得出结论：如果无限制地划分软件，那么开发软件所需的工作量将会小到忽略不计！不幸的是，其他因素开始起作用，导致这个结论是不成立的（十分遗憾）。如图 12-2 所示，开发单个软件模块的工作量（成本）的确随着模块数的增加而下降。给定同样的需求，更多的模块意味着每个模块的规模更小。然而，随着模块数量的增加，集成模块的工作量（成本）也在增加。这些特性形成了图示中的总体成本或工作量曲线。事实上，的确存在一个模块数量 M ，这个数量可以带来最小的开发成本。但是，我们缺乏成熟的技术来精确地预测 M 。

在考虑模块化的时候，图 12-2 所示的曲线确实提供了有益的指导。在进行模块化的时候，应注意保持在 M 附近，避免不足的模块化或过度的模块化问题。但是如何知道 M 的附近在哪里呢？怎样将软件划分成模块呢？回答这些问题需要理解本章后面提出的其他设计概念。

模块化设计（以及由其产生的程序）使开发工作更易于规划；可以定义和交付软件增量；更容易实施变更；能够更有效地开展测试和调试；可以进行长期维护而没有严重的副作用。

12.3.6 信息隐蔽

模块化概念面临的一个基本问题是：“应当如何分解一个软件解决方案以获得最好的模块集合？”信息隐蔽原则[Par72]建议模块应该“具有的特征是：每个模块对其他所有模块都隐蔽自己的设计决策”。换句话说，模块应该被特别说明并设计，使信息（算法和数据）都包含在模块内，其他模块无需对这些信息进行访问。

隐蔽的含义是，通过定义一系列独立的模块得到有效的模块化，独立模块之间只交流实现软件功能所必需的信息。抽象有助于定义构成软件的过程（或信息）实体。隐蔽定义并加强了对模块内过程细节的访问约束以及对模块所使用的任何局部数据结构的访问约束[Ros75]。

将信息隐蔽作为系统模块化的一个设计标准，在测试过程以及随后的

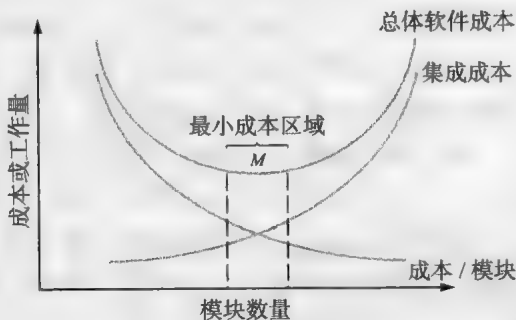


图 12-2 模块化和软件成本

提问 对于一个给定的系统，合适的模块数量是多少？

关键点 信息隐蔽的目的是将数据结构和处理过程的细节隐藏在模块接口之后。用户不需要了解模块内部的具体细节。

软件维护过程中需要进行修改时,将使我们受益匪浅。由于大多数数据和过程对软件的其他部分是隐蔽的,因此,在修改过程中不小心引入的错误就不太可能传播到软件的其他地方。

12.3.7 功能独立

功能独立的概念是关注点分离、模块化、抽象和信息隐蔽概念的直接产物。在关于软件设计的一篇里程碑性的文章中,Wirth[Wir71]和Parnas[Par72]间接提到增强模块独立性的细化技术。Stevens、Myers和Constantine[Ste74]等人在其后又巩固了这一概念。

通过开发具有“专一”功能和“避免”与其他模块过多交互的模块,可以实现功能独立。换句话说,软件设计时应使每个模块仅涉及需求的某个特定子集,并且当从程序结构的其他部分观察时,每个模块只有一个简单的接口。

人们会提出一个很合理的问题:独立性为什么如此重要?具有有效模块化(也就是独立模块)的软件更容易开发,这是因为功能被分隔而且接口被简化(考虑由一个团队进行开发时的结果)。独立模块更容易维护(和测试),因为修改设计或修改代码所引起的副作用被限制,减少了错误扩散,而且模块复用也成为可能。概括地说,功能独立是良好设计的关键,而设计又是软件质量的关键。

提问 为什么我们总是努力构造独立模块?

独立性可以通过两条定性的标准进行评估:内聚性和耦合性。内聚性显示了某个模块相关功能的强度;耦合性显示了模块间的相互依赖性。

关键点 内聚性是一个模块对于一件事情侧重程度的定性指标。

内聚性是12.3.6节说明的信息隐蔽概念的自然扩展。一个内聚的模块执行一个独立的任务,与程序的其他部分构件只需要很少的交互。简单地说,一个内聚的模块应该只完成一件事情(理想情况下)。即使我们总是争取高内聚性(即专一性),一个软件构件执行多项功能也经常是必要的和可取的。然而,为了实现良好的设计,应该避免“分裂型”构件(执行多个无关功能的构件)。

关键点 耦合性是一个模块和其他模块及外部世界连接程度的定性指标。

耦合性表明软件结构中多个模块之间的相互连接。耦合性依赖于模块之间的接口复杂性、引用或进入模块所在的点以及什么数据通过接口进行传递。在软件设计中,应当尽力得到最低可能的耦合。模块间简单的连接性使得软件易于理解并减少“涟漪效果”(ripple effect)的倾向[Ste74]。当在某个地方发生错误并传播到整个系统时,就会引起“涟漪效果”。

236

12.3.8 求精

逐步求精是一种自顶向下的设计策略,最初由Niklaus Wirth[Wir71]提出。通过连续细化过程细节层进行应用开发,通过逐步分解功能的宏观陈述(过程抽象)进行层次开发,直至最终到达程序设计语言的语句这一级。

求精实际上是一个细化的过程。该过程从高抽象级上定义的功能陈述(或信息描述)开始。也就是说,该陈述概念性地描述了功能或信息,但是没有提供有关功能内部的工作或信息内部的结构。可以在原始陈述上进行细化,随着每次细化的持续进行,将提供越来越多的细节。

建议 有一种趋势是直接进行全面详细的设计,而跳过细化步骤,这将导致错误和遗漏,并使得设计更难于评审。因此,一定要实施逐步求精。

抽象和细化是互补的概念。抽象能够明确说明内部过程和数据,但对“外部使用者”隐藏了低层细节;细化有助于在设计过程中揭示低层细节。这两个概念均有助于设计人员在设计演化中构建出完整的设计模型。

12.3.9 方面

当我们开始进行需求分析时,一组“关注点”就出现了。这些关注点“包括需求、用例、特征、数据结构、服务质量问题、变量、知识产权边界、协作、模式以及合同”[AOS07]。理想情况下,可以按某种方式组织需求模型,该方式允许分离每个关注点(需求),使得我们能够独立考虑每个关注点(需求)。然而实际上,某些关注点跨越了整个系统,从而很难进行分割。

开始进行设计时,需求被细化为模块设计表示。考虑两个需求,A和B。“如果已经选择了一种软件分解(细化),在这种分解中,如果不考虑需求A的话,需求B就不能得到满足”[Ros04],那么需求A横切需求B。

例如,考虑 www.safehomeassured.com 网站应用中的两个需求。用第9章中讨论的用例 ACS-DCV 描述需求A,设计求精将集中于那些能够使注册用户通过放置在空间中的摄像机访问视频的模块。需求B是一个通用的安全需求,要求注册用户在使用 www.safehomeassured.com 之前必须先进行验证,该需求用于 SafeHome 注册用户可使用的所有功能中。当设计求精开始的时候,A*是需求A的一个设计表示,B*是需求B的一个设计表示。因此,A*和B*是关注点的表示,且B*横切A*。

方面是一个横切关注点的表示,因此,需求注册用户在使用 www.safehomeassured.com 之前必须先进行验证的设计表示B*是 SafeHome 网站应用的一个方面。标识方面很重要,以便于在开始求精和模块化的时候,设计能够很好地适应这些方面。在理想情况下,一个方面作为一个独立的模块(构件)进行实施,而不是作为“分散的”或者和许多构件“纠缠的”软件片断进行实施[Ban06]。为了做到这一点,设计体系结构应当支持定义一个方面,该方面即一个模块,该模块能够使该关注点经过它横切的所有其他关注点而得到实施。

12.3.10 重构

很多敏捷方法(第5章)都建议一种重要的设计活动——重构,重构是一种重新组织的技术,可以简化构件的设计(或代码)而无需改变其功能或行为。Fowler[Fow00]这样定义重构:“重构是使用这样一种方式改变软件系统的过程:不改变代码(设计)的外部行为而是改进其内部结构。”

在重构软件时,检查现有设计的冗余性、没有使用的设计元素、低效的或不必要的算法、拙劣的或不恰当的数据结构以及其他设计不足,修改这些不足以获得更好的设计。例如,第一次设计迭代可能得到一个构件,表现出很低的内聚性(即执行三个功能但是相互之间仅有有限的联系)。在深思熟虑之后,设计人员可能决定将原构件重新分解为三个独立的构件,其中每个构件都表现出更高的内聚性。结果则是,软件更易于集成、测试与维护。

尽管重构的目的是以某种方式修改代码,而并不改变它的外部行为,但意外的副作用可能发生,并且也确实会发生。为此,可以使用重构工具[Soal0]自动分析变更,并“生成可用于检测行为变更的测试套件”。

引述 时常浏览一下封面,确保它不是一本关于软件设计的书,这样你会更容易读懂书中的“魔法”原理。

Bruce Tognazzini

关键点 横切关注点是系统的某个特征,它适用于许多不同的需求。

237

网络资源 重构的优秀资源可以在网站 www.refactoring.com 找到。

网络资源 大量重构模式可以在网站 <http://c2.com/cgi/wiki?RefactoringPatterns> 找到。

12.3.11 面向对象的设计概念

面向对象 (Object-Oriented, OO) 范型广泛应用于现代软件工程。附录 2 是为那些不熟悉面向对象概念 (如类、对象、继承、消息和多态等) 的读者提供的。

238

SafeHome

设计概念

[场景] Vinod 的房间, 设计建模开始。

[人物] Vinod、Jamie 和 Ed, SafeHome 软件工程团队成员; 还有 Shakira, 团队的新成员。

[对话]

(这四个团队成员上午都参加了一位本地计算机科学教授举行的名为“应用基本的设计概念”的研讨会, 他们刚从会上回来。)

Vinod: 你们从研讨会学到什么没有?

Ed: 大部分的东西我都已经知道, 但我想重温一遍总不是什么坏事。

Jamie: 我在计算机专业学习时, 从没有真正理解信息隐蔽为什么像他们说得那么重要。

Vinod: 因为……底线……这是减少错误在程序内扩散的一种技术。实际上, 功能独立做的也是同样的事。

Shakira: 我不是计算机科学专业的, 因此教授提到的很多东西对我而言都是新的。我能生成好的代码而且速度快, 我不明白这个东西为什么这么重要。

Jamie: 我了解你的工作, Shak, 你要知道, 其实你是在自然地做这些事情……这就是

为什么你的设计和编码很有效。

Shakira (微笑): 是的, 我通常的确是尽量将代码分割, 让分割后的代码关注于一件事, 保持接口简单而且有约束, 在任何可能的时候重用代码……就是这样。

Ed: 模块化、功能独立、隐蔽、模式……现在明白了。

Jamie: 我至今还记得我上的第一节编程课……他们教我们用迭代方式细化代码。

Vinod: 设计可以采用同样的方式, 你知道的。

Jamie: 我以前从未听说过的概念是“方面”和“重构”。

Shakira: 我记得她说那是用在极限编程中的。

Ed: 是的。其实它和细化并没有太大不同, 它只是在设计或代码完成后进行。我认为, 这是软件开发过程中的一种优化。

Jamie: 让我们回到 SafeHome 设计。我觉得在我们开发 SafeHome 的设计模型时, 应该将这些概念用在评审检查单上。

Vinod: 我同意。但重要的是, 我们都要能够在设计时想一想这些概念。

12.3.12 设计类

分析模型定义了一组分析类 (第 10 章), 每一个分析类都描述问题域中的某些元素, 这些元素关注用户可见的问题方面。分析类的抽象级相对较高。

当设计模型发生演化时, 必须定义一组设计类, 它们可以: (1) 通过提供设计细节对分析类进行求精, 而这些设计细节将促成类的实现; (2) 实现支持业务解决方案的软件基础设施。以下给出了五种不同类型的设计类 [Amb01], 每一种都表示了设计

提问 设计者要创建哪些类型的类?

[239]

体系结构的一个不同层次：(1) 用户接口类，定义人机交互 (Human-Computer Interaction, HCI) 所必需的所有抽象，并且经常在隐喻的环境中实施 HCI；(2) 业务域类，识别实现某些业务域元素所必需的属性和服务 (方法)，通过一个或更多的分析类进行定义；(3) 过程类，实现完整的管理业务域类所必需的低层业务抽象；(4) 持久类，用于表示将在软件执行之外持续存在的数据存储 (例如，数据库)；(5) 系统类，实现软件管理和控制功能，使得系统能够运行，并在其计算环境内与外界通信。

随着体系结构的形成，每个分析类 (第 10 章) 转化为设计表示，抽象级就降低了。也就是说，分析类使用业务域的专门用语描述数据对象 (以及数据对象所用的相关服务)。设计类更多地表现技术细节，用于指导实现。

Arlow 和 Neustadt[Arl02] 给出建议：应当对每个设计类进行评审，以确保设计类是“组织良好的”(well-formed)。他们为组织良好的设计类定义了四个特征。

完整性与充分性。设计类应该完整地封装所有可以合理预见的 (根据对类名的理解) 存在于类中的属性和方法。例如，为视频编辑软件定义的 Scene 类，只有包含与创建视频场景相关的所有合理的属性和方法时，它才是完整的。充分性确保设计类只包含那些“对实现该类的目的是足够”的方法，不多也不少。

提问 什么才是“组织良好的”设计类？

原始性。和一个设计类相关的方法应该关注于实现类的某一个服务。一旦服务已经被某个方法实现，类就不应该再提供完成同一事情的另外一种方法。例如，视频编辑软件的 VideoClip 类，可能用属性 start-point 和 end-point 指定剪辑的起点和终点 (注意，加载到系统的原始视频可能比要用的部分长)。方法 setStartPoint() 和 setEndPoint() 为剪辑提供了设置起点和终点的唯一手段。

高内聚性。一个内聚的设计类具有小的、集中的职责集合，并且专注于使用属性和方法来实现那些职责。例如，视频编辑软件的 VideoClip 类可能包含一组用于编辑视频剪辑的方法。只要每个方法只关注于和视频剪辑相关的属性，内聚性就得以维持。

低耦合性。在设计模型内，设计类之间相互协作是必然的。但是，协作应该保持在一个可以接受的最小范围内。如果设计模型高度耦合 (每一个设计类都和其他所有的设计类有协作关系)，那么系统就难以实现、测试，并且维护也很费力。通常，一个子系统内的设计类对其他子系统内的类应仅有有限的了解。该限制被称作 Demeter 定律 [Lie03]，该定律建议一个方法应该只向周边类中的方法发送消息。^①

[240]

SafeHome 将分析类细化为设计类

[场景] Ed 的房间，开始进行设计建模。

模型的细化。)

[人物] Vinod 和 Ed, SafeHome 软件工程技术成员。

Ed: 你还记得 FloorPlan 类吗？这个类用作监视和住宅管理功能的一部分。

[对话]

(Ed 正进行 FloorPlan 类的设计工作 (参考 10.3 节的讨论以及图 10-2)，并进行设计

Vinod (点头): 是的，我好像想起来我们把它用作住宅管理 CRC 讨论的一部分。

Ed: 确实如此。不管怎样，我们要对设计

^① Demeter 定律的一种非正式表述是：“每个单元应该只和它的朋友谈话，不要和陌生人谈话。”

进行细化，希望显示出我们将如何真正地实现 FloorPlan 类。我的想法是把它实现为一组链表（一种特定的数据结构）。像这样……我必须细化分析类 FloorPlan（图 10-2），实际上，是它的一种简化。

Vinod：分析类只显示问题域中的东西，也就是说，在电脑屏幕上实际显示的、最终用户可见的那些东西，对吗？

Ed：是的。但对于 FloorPlan 设计类来说，我已经开始添加一些实现中特有的东西。需要说明的是 FloorPlan 是段（Segment 类）的聚集，Segment 类由墙段、窗户、

门等的列表组成。Camera 类和 FloorPlan 类协作，这很显然，因为在平面图中可以有很多摄像机。

Vinod：咳，让我们看看新的 FloorPlan 设计类图。

（Ed 向 Vinod 展示图 12-3。）

Vinod：好的，我看出来你想做什么了。这样你能够很容易地修改平面图，因为新的东西可以在列表（聚集）中添加或删除，而不会有任何问题。

Ed（点头）：是的，我认为这样是可以的。

Vinod：我也赞同。

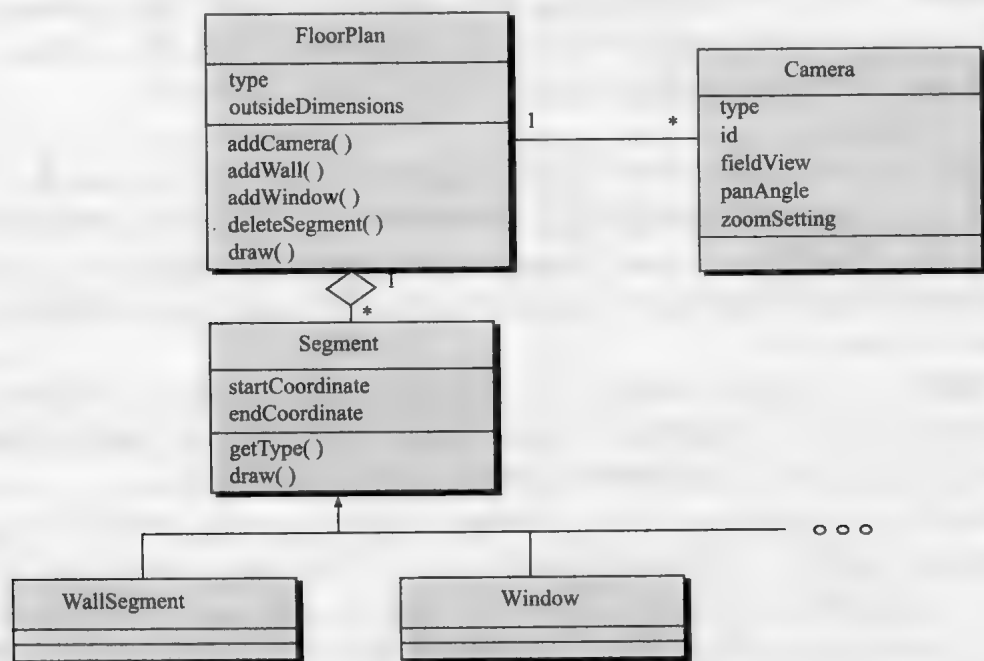


图 12-3 FloorPlan 的设计类和类的复合聚集

12.3.13 依赖倒置

许多旧一些的软件体系结构是层次结构。在体系结构的顶层，“控制”构件依赖于较低层的“工作者”构件，以完成各种内聚任务。比如，考虑一个具有三个组件的简单程序，该程序的目的是读取键盘的按键并将结果输出到打印机。控制模块 C 需要协调另外两个模块——按键读取模块 R 和写入打印机模块 W。

由于控制模块 C 高度依赖于 R 和 W，因此程序设计是耦合的。为了消除这种依赖，“工作者”模块 R 和 W 应当通过抽象由控制模块 C 调用。在面向对象的软件工程中，抽象被作为抽象类 R* 和 W* 以获得实现。然

提问 依赖倒置的原则是什么？

后, 这些抽象类可以用来调用执行读写功能的工作者类。因此, 一个 copy 类 C 调用抽象类 R* 和 W*, 抽象类指向合适的工作者类 (例如, R* 类可能指向一个环境中 keyboard 类的 read() 操作和另一个环境中 sensor 类的 read() 操作)。这种方法降低了耦合性, 并提高了设计的可测性。

前段中讨论的例子可以与依赖倒置原则一概而论 [Obj10], 该原则这样描述: 高层模块 (类) 不应当 (直接) 依赖于低层模块, 两者都应当依赖于抽象。抽象不应当依赖于细节, 细节应当依赖于抽象。

12.3.14 测试设计

到底应当先开始软件设计还是测试用例设计, 这是一个争论不休的鸡与蛋的问题。Rebecca Wirfs-Brock [Wir09] 写道:

测试驱动开发的倡导者们在编写任何其他代码之前先编写测试代码。他们谨记 Peter 的信条——测试要快, 失败要快, 调整要快。他们以一种简短而快速的方式实施周期演变, 编写测试代码——测试未通过——编写足够的代码以通过测试——测试通过, 在这一过程中, 测试指导他们的设计。

引述 测试要快, 失败要快, 调整要快。

Tom Peters

242

但如果先开始进行设计, 那么设计 (以及编码) 必须带有一些接缝。所谓接缝, 即详细设计中的一些位置, 在这些位置可以 “插入一些测试代码, 这些代码可用以探测运行中软件的状态”, 以及 “将待测试的代码从它的产生环境中分离出来, 以便在受控的测试环境中执行这些代码” [Wir09]。

有时候, 接缝也被称为 “测试沟”, 它们必须被有意识地在构件级进行设计。为了实现这一点, 设计者必须考虑将用于演练构件的测试。正如 Wirf-Brock 所述: “简言之, 需要提供适当的测试启示——以一种方式将设计分解为若干因素, 测试代码可以询问并控制运行中的系统。”

12.4 设计模型

可以从两个不同的维度观察设计模型, 如图 12-4 所示。过程维度表示设计模型的演化, 设计任务作为软件过程的一部分被执行。抽象维度表示详细级别, 分析模型的每个元素转化为一个等价的设计, 然后迭代求精。参考图 12-4, 虚线表示分析模型和设计模型之间的边界。在某些情况下, 分析模型和设计模型之间可能存在明显的差异; 而有些情况下, 分析模型慢慢地融入设计模型而没有明显的差异。

设计模型的元素使用了很多 UML 图^①, 有些 UML 图在分析模型中也会用到。差别在于这些图被求精和细化为设计的一部分, 并且提供了更多具体的实施细节, 突出了体系结构的结构和风格、体系结构中的构件、构件之间以及构件和外界之间的接口。

关键点 设计模型有 4 个主要元素: 数据、体系结构、构件和接口。

243

然而, 要注意到, 沿横轴表示的模型元素并不总是顺序开发的。大多数情况下, 初步的体系结构设计是基础, 随后是接口设计和构件级设计 (通常是并行进行)。通常, 直到设计全部完成后才开始部署模型的工作。

① 附录 1 提供了基本 UML 概念和符号的使用手册。

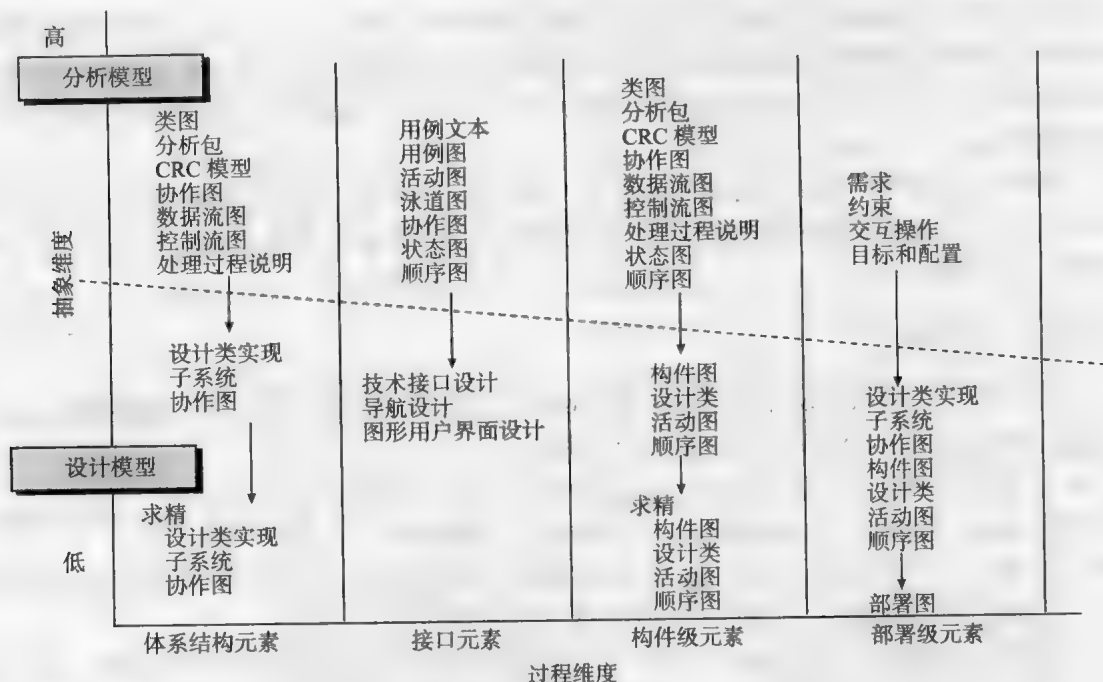


图 12-4 设计模型的维度

在设计过程中的任何地方都可以应用设计模式（第 16 章）。这些模式能够使设计人员将设计知识应用到他人已经遇到并解决了的特定领域问题中。

12.4.1 数据设计元素

和其他软件工程活动一样，数据设计（有时也称为数据体系结构）创建了在高抽象级上（以客户或用户的数据观点）表示的数据模型和信息模型。之后，数据模型被逐步求精为特定实现的表示，亦即计算机系统能够处理的表示。在很多软件应用中，数据体系结构对于必须处理该数据的软件的体系结构将产生深远的影响。

数据结构通常是软件设计的重要部分。在程序构件级，数据结构设计以及处理这些数据的相关算法对于创建高质量的应用程序是至关重要的。在应用级，从数据模型（源自于需求工程）到数据库的转变是实现系统业务目标的关键。在业务级，收集存储在不同数据库中的信息并重新组织为“数据仓库”要使用数据挖掘或知识发现技术，这些技术将会影响到业务本身的成功。在每一种情况下，数据设计都发挥了重要作用。第 13 章将更详细地讨论数据设计。

12.4.2 体系结构设计元素

软件的体系结构设计等效于房屋的平面图。平面图描绘了房间的整体布局，包括各房间的尺寸、形状、相互之间的联系，能够进出房间的门窗。平面图为我们提供了房屋的整体视图；而体系结构设计元素为我们提供了软件的整体视图。

引述 提出“设计是否是必要的”或“能否负担得起”这样的问题非常离题，因为设计是不可避免的。不是好的设计就是坏的设计，根本不可能不要设计。

Douglas Martin

关键点 在体系结构（应用）级，数据设计关注文件或数据库；在构件级，数据设计考虑实现局部数据对象所需的数据结构。

体系结构模型 [Sha96] 从以下三个来源导出：(1) 关于将要构建的软件的应用域信息；(2) 特定的需求模型元素，如数据流图或分析类、现有问题中它们的关系和协作；(3) 可获得的体系结构风格（第13章）和模式（第16章）。

体系结构设计元素通常被描述为一组相互联系的子系统，且常常从需求模型中的分析包中派生出来。每个子系统有其自己的体系结构（如图形用户界面可能根据之前存在的用户接口体系结构进行了结构化）。体系结构模型特定元素的导出技术将在第13章中介绍。

12.4.3 接口设计元素

软件的接口设计相当于一组房屋的门、窗和外部设施的详细绘图（以及规格说明）。门、窗、外部设施的详细图纸（以及规格说明）作为平面图的一部分，大体上告诉我们：事件和信息如何流入和流出住宅以及如何如何在平面图的房间内流动。软件接口设计元素描述了信息如何流入和流出系统，以及被定义为体系结构一部分的构件之间是如何通信的。

接口设计有三个重要的元素：(1) 用户界面（User Interface, UI）；(2) 和其他系统、设备、网络、信息生成者或使用者的外部接口；(3) 各种设计构件之间的内部接口。这些接口设计元素能够使软件进行外部通信，还能使软件体系结构中的构件之间进行内部通信和协作。

UI 设计（越来越多地被称作可用性设计）是软件工程中的主要活动，这会在第15章中详细地考虑。可用性设计包含美学元素（例如，布局、颜色、图形、交互机制）、人机工程元素（例如，信息布局、隐喻、UI 导航）和技术元素（例如，UI 模式、可复用构件）。通常，UI 是整个应用体系结构内独一无二的子系统。

外部接口设计需要发送和接收信息实体的确定信息。在所有情况下，这些信息都要在需求工程（第8章）过程中进行收集，并且在接口[⊖]设计开始时进行校验。外部接口设计应包括错误检查和适当的安全特征检查。

内部接口设计和构件级设计（第14章）紧密相关。分析类的设计实现呈现了所有需要的操作和消息传递模式，使得不同类的操作之间能够进行通信和协作。每个消息的设计必须提供必不可少的信息传递以及所请求操作的特定功能需求。

在有些情况下，接口建模的方式和类所用的方式几乎一样。在 UML 中，接口如下定义 [OMG03a]：“接口是类、构件或其他分类符（包括子系统）的外部可见的（公共的）操作说明，而没有内部结构的规格说明。”更简单地说，接口是一组描述类的部分行为的操作，并提供了这些操作的访问方法。

例如，SafeHome 安全功能使用控制面板，控制面板允许户主控制安全功能的某些方面。在系统的高级版本中，控制面板的功能可能会通过移

引述 你可以在绘图桌上使用橡皮或在建筑工地现场使用大铁锤。

Frank Lloyd Wright

引述 与良好的设计相比，公众更熟悉拙劣的设计。实际上，公众更习惯拙劣的设计，因为生活就是如此。新的有危险，而旧的更让人安心。

Paul Rand

关键点 接口设计元素有三部分：用户接口、系统和外部应用的接口、应用系统内部构件之间的接口。

引述 现在的每一样东西以后都会消失。稍微放松一下，再返回到你的工作中，你的判断将更可靠。因为离开一定距离，工作看起来更小，更容易远瞰，更容易发现和谐和比例的缺失。

Leonardo Da Vinci

245

⊖ 接口特征可能随时间变化。因此，设计者应当确保接口的规格说明是准确且完整的。

动平台（例如智能手机或平板电脑）实现。

ControlPanel 类（图 12-5）提供了和键盘相关的行为，因此必须实现操作 readKeyStroke() 和 decodeKey()。如果这些操作提供给其他类（在此例中是 Tablet 和 SmartPhone），定义如图 12-5 所示的接口是非常有用的。名为 KeyPad 的接口表示为 <<interface>> 构造型（stereotype），或用一个带有标识且用一条线和类相连的小圆圈表示，定义接口时并没有实现键盘行为所必需的属性和操作集合。

网络资源 有关 UI 设计非常有用的信息可以在 www.useit.com 找到。

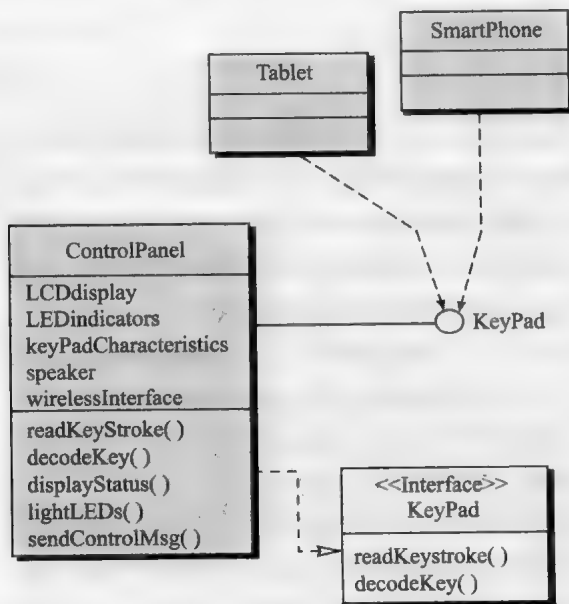


图 12-5 ControlPanel 的接口表示

[246]

带有三角箭头的虚线（图 12-5）表示 ControlPanel 类提供了 KeyPad 操作以作为其行为的一部分。在 UML 中，这被称为实现。也就是说，ControlPanel 行为的一部分将通过实现 KeyPad 操作来实现。这些操作将被提供给那些访问该接口的其他类。

引述 在设计傻瓜级的东西时，人们常犯的一个错误是低估傻瓜的聪明。

Douglas Adams

12.4.4 构件级设计元素

软件的构件级设计相当于一个房屋中每个房间的一组详图（以及规格说明）。这些图描绘了每个房间内的布线和管道、电器插座和墙上开关、水龙头、水池、淋浴、浴盆、下水道、壁橱和储藏室的位置，以及房间相关的任何其他细节。

软件的构件级设计完整地描述了每个软件构件的内部细节。为此，构件级设计为所有局部数据对象定义数据结构，为所有在构件内发生的处理定义算法细节，并定义允许访问所有构件操作（行为）的接口。

在面向对象的软件工程中，使用 UML 图表现的一个构件如图 12-6 所示。图中表示的构件名为 SensorManagement（SafeHome 安全功能的一部分）。虚线箭头连接了构件和名为 Sensor 的类。SensorManagement 构件完成所有和 SafeHome 传感器相关的功能，包括监控和配置传感器。第 14 章将进一步讨论构件图。

引述 细节并不仅仅是细节，细节构成设计。

Charles Eames

构件的设计细节可以在很多不同的抽象级进行建模。UML 活动图可用来表示处理逻辑，构件详细的过程流可以使用伪代码表示（类似编程语言的表示方法，在第 14 章讨论），也可以使用一些图形（例如流程图或盒图）来表示。算法结构遵守结构化编程的规则（即一套约束程序构造）。基于待处理数据对象的特性所选择的数据结构，通常会使用伪代码或程序语言进行建模，以便将之用于实施。



图 12-6 UML 构件图

247

12.4.5 部署级设计元素

部署级设计元素指明软件功能和子系统将如何在支持软件的物理计算环境内进行分布。例如，SafeHome 产品元素被配置在三种主要的计算环境内运行——基于住宅的 PC、SafeHome 控制面板和位于 CPI 公司的服务器（提供基于 Internet 的系统访问）。此外，移动平台也可以提供有限的功能。

在设计过程中，开发的 UML 部署图以及随后的细化如图 12-7 所示。图中标识出了每个计算元素中还有子系统（功能）。例如，个人计算机中有完成安全、监视、住宅管理和通信功能的子系统。此外，还设计了一个外部访问子系统，以管理外界资源对 SafeHome 系统的访问。每个子系统需要进行细化，用以说明该子系统所实现的构件。

关键点 部署图刚开始使用描述符形式，粗略描述部署环境。后来使用实例形式，明确描述配置的元素。

如图 12-7 所示，图中使用了描述符形式，这意味着部署图表明了计算环境，但并没有明确地说明配置细节。例如，“个人计算机”并没有进一步地明确它是一台 Mac PC、一台基于 Windows 的 PC、Linux 系统还是带有相关操作系统的移动平台。在设计后续阶段或构建开始时，需要用实例形式重新为部署图提供这些细节，明确每个实例的部署（专用的称为硬件配置）。

248

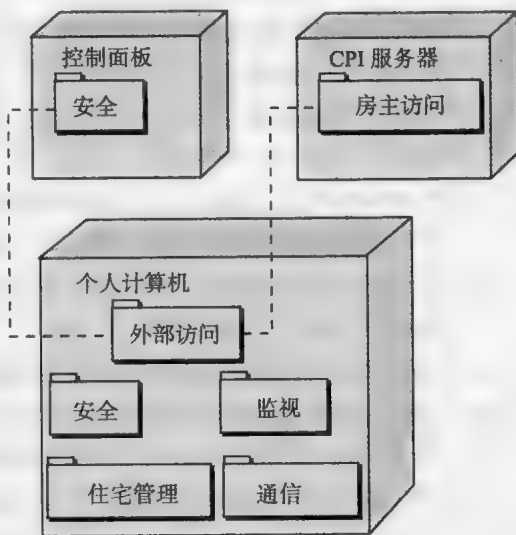


图 12-7 UML 部署图

12.5 小结

当第一次需求工程得到结论时，软件设计便开始了。软件设计的目的是应用一系列原则、概念和实践，以引导高质量的系统或产品开发。设计的目标是创建软件模型，该模型将正确地实现所有的客户需求，并为软件用户带来愉悦的感受。软件设计人员必须从大量可供选择的设计中筛选并确定一个解决方案，该方案最能满足项目利益相关者的需要。

设计过程从软件的“宏观”视图向微观视图转移，后者定义了实现系统所必需的细节。设计过程开始时关注于体系结构，然后定义子系统、建立子系统之间的通信机制、识别构件、制定每个构件的详细说明，另外还要设计外部接口、内部接口和用户接口。

设计概念在软件工程刚开始的 60 年内不断发展。这些概念描述了计算机软件的属性

(而并不应考虑所选择的软件工程过程)、描述所使用的设计方法或所使用的编程语言。实质上,设计概念强调了:(1)抽象的必要性,它提供了一种创造可重用软件构件的方法;(2)体系结构的重要性,它使得人们能够更好地理解系统整体结构;(3)基于模式的工程的有益性,它将已证明的能力用于软件设计;(4)关注点分离和有效模块化的价值,它们使得软件更易理解、更易测试以及更易维护;(5)信息隐蔽的直接作用,当错误发生时,它能够减少负面影响的传播;(6)功能独立的影响,它是构建有效模块的标准;(7)求精作为一种设计方法的作用;(8)横切系统需求方面的考虑;(9)重构的应用,目的是优化已导出的设计;(10)面向对象的类和与类相关特征的重要性;(11)使用抽象降低构件之间耦合的需要;(12)测试设计的重要性。

249

设计模型包含四种不同的元素。随着每个元素的开发,逐渐形成更全面的设计视图。体系结构元素使用各种信息以获得软件、软件子系统和构件的完整的结构表示,这些信息来自于应用域、需求模型以及模式和风格的分类。接口设计元素为外部和内部的接口以及用户接口建模。构件级元素定义体系结构中的每一个模块(构件)。最后,部署级设计元素划分体系结构、构件和容纳软件的物理配置的接口。

习题与思考题

- 12.1 当你“编写”程序时是否会设计软件?软件设计和编码有什么不同?
- 12.2 如果软件设计不是程序(它肯定不是),那么它是什么?
- 12.3 如何评估软件设计的质量?
- 12.4 查看设计任务集,在任务集中的什么地方对质量进行评估?评估是如何完成的?如何达到12.2.1节中讨论的质量属性?
- 12.5 举三个数据抽象和能用来控制数据的过程抽象的例子。
- 12.6 用你自己的话描述软件体系结构。
- 12.7 为你每天都能遇到的东西(例如家用电器、汽车、设备)推荐一个设计模式,简要地描述该模式。
- 12.8 用你自己的语言描述关注点分离。分而治之的方法有时候不合适吗?这种情况对模块化的观点有多大的影响?
- 12.9 应在什么时候把模块设计实现为单块集成软件?如何实现?性能是实现单块集成软件的唯一理由吗?
- 12.10 讨论作为有效模块化属性的信息隐蔽概念和模块独立性概念之间的联系。
- 12.11 耦合性的概念如何与软件可移植性相关联?举例支持你的论述。
- 12.12 应用“逐步求精方法”为下列一个或多个程序开发三种不同级别的过程抽象:(1)开发一个支票打印程序,给出金额总数,并按支票的常规要求给出大写金额数;(2)为某个超越方程迭代求解;(3)为操作系统开发一个简单的任务调度算法。
- 12.13 考虑需要实现汽车导航功能(GPS)、手持通信设备的软件。描述两个或三个要表示的横切关注点。讨论如何将其中一个关注点作为方面来表示。
- 12.14 “重构”意味着迭代地修改整个设计吗?如果不是,它意味着什么?
- 12.15 用你自己的语言描述什么是依赖倒置?
- 12.16 测试设计为什么很重要?
- 12.17 简要描述设计模型的四个元素。

250

扩展阅读与信息资源

Donald Norman 编写了三本书:《Emotional Design: We love(or hate) Everyday Things》(Basic Books, 2005);《The Design of Everyday Things》(Doubleday, 1990) 以及《The Psychology of Everyday Things》(HarperCollins, 1988)。这三本书已经成为设计学中的经典著作,任何人想设计人类使用的任何东西,都“必须”阅读这些著作。Adams 的著作(《Conceptual Blockbusting》, 4th ed., Addison-Wesley, 2001)对那些希望拓宽思路的设计人员极为重要。最后, Polya 在其编写的一本经典著作(《How to Solve It》, 2nd ed., Princeton University Press, 1988)中提供了通用的问题解决流程,在软件设计人员面对复杂问题时能够提供帮助。

Hanington 和 Martin (《Universal Methods of Design: 100 ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions》, Rockport, 2012 和《Universal Principles of Design: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design》, 2nd ed., Rockport, 2010)讨论了通常的设计原则。

遵循同样的传统, Winograd 等人(《Bringing Design to Software》, Addison-Wesley, 1996)讨论了成功与不成功的软件设计及其理由。Wixon 和 Ramsey 编写了一本令人着迷的书(《Field Methods Casebook for Software Design》, Wiley, 1996),书中建议使用领域搜索方法(和人类学家所使用的那些方法非常类似)理解最终用户是如何工作的,然后设计满足用户需要的软件。Holtzblatt (《Rapid Contextual Design: A How-to Guide to Key Techniques for User-Center Design》, Morgan Kaufman, 2004)以及 Beyer 和 Holtzblatt (《Contextual Design: A Customer-Centered Approach to Systems Designs》, Academic Press, 1997)提供了软件设计的另一种视图,即将客户/用户集成到软件设计流程的各个方面。Bain (《Emergent Design》, Addison-Wesley, 2008)将模式、重构和测试驱动的开发方法结合到有效的设计方法中。

Otero(《Software Engineering Design:Theory and Practice》, Auerbach, 2012)、Venit 和 Drake (《Prelude to Programming: Concepts and Design》, 5th ed., Addison-Wesley, 2010)、Fox (《Introduction to software Engineering Design》, Addison-Wesley, 2006)以及 Zhu (《Software Design Methodology》, Butterworth-Heinemann, 2005)介绍了软件工程中设计的综合性处理。McConnell (《Code Complete》, 2nd ed., Microsoft Press, 2004)对高质量计算机软件的实践方面进行了精彩的论述。Robertson (《Simple Program Design》, 5th ed., Course Technology, 2006)介绍性地论述了软件设计,这对初学者很有帮助。Budgen (《Software Design》, 2nd ed., Addison-Wesley, 2004)介绍了大量流行的软件设计方法,并对它们进行了对比。Fowler 和他的同事(《Refactoring: Improving the Design of Existing Code》, Addison-Wesley, 1999)讨论了软件设计增量优化的技术。Rosenberg 和 Stevens (《Use Case Driven Object Modeling with UML》, Apress, 2007)讨论了以用例为基础的面向对象的设计开发。

从 Free-man 和 Wasserman (《Software Design Techniques》, 4th ed., IEEE, 1983)编辑的文集中,可以一览软件设计的精彩发展历史。这本教程中转载了许多经典的论文,这些论文已经奠定了软件设计当前发展趋势的基础。Card 和 Glass (《Measuring Software Design Quality》, Prentice-Hall, 1990)从技术和管理两个角度介绍并思考了软件质量的度量。

网上有关于软件设计的大量信息资源。在 SEPA 网站 www.mhhe.com/pressman 上可以找到与软件设计以及设计工程相关的最新参考文献。

体系结构设计

要点浏览

概念: 体系结构设计表示建立计算机系统所需的数据结构和程序构件。它需要考虑系统采取的体系结构风格、系统组成构件的结构和属性以及系统中所有体系结构构件之间的相互关系。

人员: 尽管软件工程师能够设计数据和体系结构,但在构造大型复杂系统时,这项工作往往由专家来完成。数据库或者数据仓库设计者为系统创建数据体系结构。“系统架构师”根据软件需求分析中导出的需求选择合适的体系结构风格。

重要性: 没有图纸就不要试图盖房子,难道不是吗?同样,也不能通过勾画房子的管道布局而开始绘制房屋的蓝图。在开始考虑细节之前,需要关注“宏观”视图,即房子本身。这就是体系结构设

计需要做的事情——它提供“宏观”视图,并确保可以正确理解该视图。

步骤: 体系结构设计始于数据设计,然后导出系统体系结构的一个或多个表示。对可选的体系结构风格或模式进行分析,得出最适于客户需求和质量属性的结构。方案一旦选定,就需要使用体系结构设计方法对体系结构进行细化。

工作产品: 在体系结构设计过程中,要创建一个包括数据体系结构和程序结构的体系结构模型。此外,还需描述构件的属性以及关系(交互作用)。

质量保证措施: 在每个阶段,都要对软件设计的工作产品进行评审,以确保工作产品与需求之间以及工作产品彼此之间的清晰性、正确性、完整性和一致性。

设计通常被描述为一个多步过程,该过程从信息需求中综合出数据和程序结构的表示、接口特征和过程细节。Freeman[Fre80]扩展了该描述:

设计活动关注如何做出重要决策,而且往往是结构性的。设计和编程都关注抽象信息表示和处理顺序,但在详细程度上,两者俨然不同。设计是构建内聚的、良好规划的程序表示,它关注高层各部分之间的相互关系和低层所包括的逻辑操作。

正如第 12 章所提到的,设计是由信息驱动的。软件设计方法是通过仔细考虑分析模型的三个域而得到的。数据、功能和行为这三个域是创建软件设计的指南。

本章将介绍建立设计模型的数据和体系结构层的“内聚的、规划良好的表示”所需的方法。目标是提供一种导出体系结构设计的系统化方法,而体系结构设计是构建软件的初始蓝图。

关键概念

- 敏捷和体系结构原型
- 体系结构决策
- 体系结构描述语言
- 体系结构描述
- 体系结构设计
- 体系结构类型
- 体系结构模式
- 体系结构风格
- 体系结构
- 体系结构一致性检查
- 体系结构细化

13.1 软件体系结构

Shaw 和 Garlan[Sha96] 在他们划时代的著作中以如下方式讨论了软件的体系结构：

从第一个程序被划分成模块开始，软件系统就有了体系结构。同时，程序员已经开始负责模块间的交互和模块装配的全局属性。从历史的观点看，体系结构隐含了不同的内容——实现的偶然事件或先前的遗留系统。优秀的软件开发人员经常采用一个或者多个体系结构模式作为系统组织策略，但是他们只是非正式地使用这些模式，并且在最终系统中没有将这些模式清楚地体现出来。

如今，有效的软件体系结构及其明确的表示和设计已经成为软件工程领域的主导主题。

13.1.1 什么是体系结构

当你考虑建筑物的体系结构时，脑海中会出现很多不同的属性。在最简单的层面上，会考虑物理结构的整体形状。但在实际中，体系结构还包含更多的方面。它是各种不同建筑构件集成为一个有机整体的方式；是建筑融入所在环境并与相邻的其他建筑相互吻合的方式；是建筑满足既定目标和满足主人要求的程度；是对结构的一种美学观感（建筑的视觉效果），以及纹理、颜色和材料结合在一起创建外观和内部“居住环境”的方式；是很多微小的细节——灯具、地板类型、壁挂布置等的设计。总而言之，它是艺术。

体系结构也可以是其他的东西。它是“数以千计的或大或小的决定”[Tyr05]。其中一些决定是设计初期做出的，并可能会对所有其他设计行为产生深刻的影响。另外一些决定一直推迟到后来才做出，因此消除了过分限制性的制约因素，而这些制约因素可能会导致拙劣的体系结构风格。

但是，什么是软件体系结构呢？Bass、Clements 和 Kazman[Bas03] 对于这个难懂的概念给出了如下定义。

程序或计算系统的软件体系结构是指系统的一个或者多个结构，它包括软件构件、构件的外部可见属性以及它们之间的相互关系。

体系结构并非可运行的软件。确切地说，它是一种表达，使你能够：（1）对设计在满足既定需求方面的有效性进行分析；（2）在设计变更相对容易的阶段，考虑体系结构可能的选择方案；（3）降低与软件构建相关的风险。

该定义强调了“软件构件”在任意体系结构表示中的作用。在体系结构设计环境中，软件构件可能会像程序模块或者面向对象的类那样简单，但也可能扩充到包含数据库和能够完成客户与服务器网络配置的“中间件”。构件的属性是理解构件之间如何相互作用的必要特征。在体系结构层次上，不会详细说明内部属性（如算法的细节）。构件之间的关系可以像从一个模块对另一个模块进行过程调用那样简单，也可以像数据库访问协议那样复杂。

软件工程界（如 [Kaz03]）的一些成员对导出软件体系结构（称为“体系结构设计”）和导出软件设计这两种行为作了区分。正如之前版本的一位评论者指出：

“体系结构”和“设计”这两个术语之间有明显的不同。设计是体系结构的一个实例，类似于对象是类的实例一样。例如，考虑“客户-服务器”体系结构，我们可以使用 Java

引述 系统的体系结构是一个关于系统形式和结构的综合框架，包括系统构件和构件的整合。

Jerrold Grochow

关键点 软件体系结构必须对系统结构以及数据和过程构件相互协作的方式进行建模。

引述 体系结构忙中建，闲时悔。

Barry Boehm

平台 (Java EE) 或者 Microsoft 平台 (.NET 框架), 选择基于该体系结构的多种不同的实现方式设计一个网络中心软件系统。即使是同一个体系结构, 也可能产生多种基于该体系结构的设计。因此, 不能把“体系结构”和“设计”混为一谈。

尽管本书同意软件设计是特定软件体系结构的实例, 但元素和结构作为体系结构的一部分, 仍是每个设计的根本。设计开始于对体系结构的思考。

13.1.2 体系结构为什么重要

在一本关于软件体系结构的书中, Bass 和他的同事 [Bas03] 给出了软件体系结构之所以重要的三个关键原因:

- 软件体系结构提供了一种表示, 有助于对计算机系统开发感兴趣的所有利益相关者开展交流。
- 体系结构突出了早期的设计决策, 这些决策对随后所有的软件工作有深远的影响。
- 体系结构“构建了一个相对小的、易于理解的模型, 该模型描述了系统如何构成以及其构件如何一起工作” [Bas03]。

体系结构设计模型和包含在其中的体系结构模式都是可以传递的, 也就是说, 体系结构的类型、风格和模式 (13.2 ~ 13.6 节) 可以应用于其他系统的设计, 并且表示了一组抽象, 使得软件工程师能以可预见的方式描述体系结构。

网络资源 可以在 <http://www.ewi.ta.com/links/softwareArchitectureLinks.htm> 获得许多软件体系结构站点的可用链接。

关键点 体系结构模型提供了系统的 Gestalt 视图, 允许软件工程师在总体上审查它。

13.1.3 体系结构描述

对于体系结构这个词的意义, 每个人都会有一种理解。因为, 不同的参与者会从不同的角度理解体系结构, 这个角度是由不同的关注点驱动的。这就意味着体系结构描述实际上是一组体现系统不同视图的工作产品。

Smolander、Rossi 和 Purao [Smo08] 提出了多个比喻, 从不同的角度来说明同一体系结构, 以便不同的参与者能更好地理解软件体系结构这个术语。对于那些编写程序来实现系统的参与者来说, 他们更习惯于将其称为蓝图。开发人员将体系结构描述为一种传递准确信息的工具, 从体系结构分析师到设计师乃至生产系统构件的软件工程师之间都可以传递这种信息。语言则侧重于将其视为不同角色人群之间进行沟通的工具, 那些具有较高客户视野的参与者 (如管理者、市场专业人士等) 较偏向于这个角度。因为体系结构的描述为后续的协商奠定了基础, 特别是在确定系统边界方面, 故而它应该是简洁易懂的。

体系结构又可比喻为在对诸如成本、可用性、可维护性、性能等属性进行权衡取舍后做出的决策, 这些属性都会对系统设计产生重大影响。利益相关者 (如项目经理) 需基于体系结构的决策来分配项目资源和工作任务。这些决策可能会影响任务的排序和软件团队的组成。文献比喻对过往已经搭建完成的体系结构形成方案文档, 可以支持构建产品, 并且将产品设计思想传递给软件维护人员, 同样也支持关心构件和设计重用的项目相关人员。

软件体系的体系结构描述也必须展示出以上这些不同视角所组合的特征。Tyree 和 Akerman [Tyr05] 注意到了这一点, 他们写道:

开发人员想要对设计进行明确、果断的指导; 客户想要对必然发生的环境变化进行清晰的理解, 以及确保体系结构将满足他们的业务需要; 而体系结构设计师想要对体系结构的关

键方面进行清晰而深入的理解。

这里每一个“想要”的东西都反映了不同视点上的不同视角。

IEEE 计算机学会提出了 IEEE-Std-1471-2000, 即“密集型软件系统体系结构描述的推荐实践做法”(Recommended Practice for Architectural Description of Software-Intensive System)[IEE00], 目标如下:(1) 建立软件体系结构设计过程中使用的概念性框架和词汇表;(2) 提供表示体系结构描述的详细准则;(3) 鼓励良好的体系结构设计实践。体系结构描述(Architectural Description, AD)展示了多个视图, 每个视图都是“从一组参与者关注点的角度观察的整个系统的一种表示”。

13.1.4 体系结构决策

视图作为体系结构描述的一部分, 解决一个特定利益相关者的关注点。为了开发每个视图(和作为整体的体系结构描述), 系统体系结构设计师会考虑多种可选方案, 并最终就最能满足关注点的特定的体系结构特征做出决策。因此, 可以将体系结构决策本身看作体系结构的一种视图。通过理解做出体系结构决策的缘由可以深刻洞悉系统的结构以及系统与利益相关者关注点的一致性。

作为一名系统体系结构设计师, 可以使用下面推荐的模板记录每个主要的决策。通过记录, 设计师为他的工作提供了逻辑依据, 并且还可以建立历史记录, 该记录在需要进行设计修改时可能有用。

Grady Booch [Boo11a] 写道, 刚开始启动一项创新性的产品开发时, 软件工程师们往往会感觉到被迫一头扎进去, 他们搭建原材料、修复问题、改进现有的处理方式, 然后重复着这些过程。但是经过几轮反复之后, 他们意识到有必要将所选用的体系结构以及与之相关的决策清晰地表达出来。在构建一项新产品之前, 还无法预见到何为正确的选择。然而, 当开发者们对新的产品原型进行现场测试后, 觉得该项体系结构方案值得重复使用时, 那么针对此类产品的主导设计^①就开始显现了。如果不将工作中的成功与失败记录下来, 软件工程师们就很难决定何时需设计新的方案, 何时采用现有的体系结构方案。

256

信息栏 体系结构决策描述模板

每个主要的体系结构决策都可以被记录在案, 以便以后评审, 评审由想要理解已提出的体系结构描述的利益相关者进行。这里给出的是 Tyree 和 Ackerman [Tyr05] 提出模板的修改和缩略版本。

设计问题: 描述将要解决的体系结构设计问题。

解决方案: 陈述所选择的解决设计问题的方法。

分类: 指定问题和解决方案陈述的分类(例如, 数据设计、内容结构、构件结构、集成、简要说明)。

假设: 指出任何有助于制定决策的假设。

约束: 指定任何有助于制定决策的环境约束(例如, 技术标准、可用的模式、项目相关问题)。

候选方案: 简要描述所考虑的体系结构设计候选方案, 并描述为什么要摒弃这些方案。

① 主导设计是指一种创新的软件体系结构或方法在市场上经过一段时间的成功适应和使用后而成为了工业标准。

争论: 陈述你为什么选择了这种解决方案而不是其他的候选方案。

意义: 指出制定决策对设计的影响。选择方案如何影响其他的体系结构设计问题?

解决方案会在某种程度上约束设计吗?

相关决策: 其他记录的决策和该决策有什么相关性?

相关关注点: 其他需求和该决策有什么相关性?

工作产品: 指出在体系结构描述中, 决策会在哪里体现出来。

注释: 参考可用来制定决策的其他团队的备忘录或文档。

13.2 体系结构类型

尽管体系结构设计的基本原则适用于所有类型的体系结构, 但对于需要构建的结构, 体系结构类型 (genre) 经常会规定特定的体系结构方法。在体系结构设计环境中, 类型隐含了在整个软件领域中的一个特定类别。在每种类别中, 会有很多的子类别。例如, 在建筑物类型中, 会有以下几种通用风格: 住宅房、单元楼、公寓、办公楼、工厂厂房、仓库等。在每一种通用风格中, 也会运用更多的具体风格 (13.3 节)。每种风格有一个结构, 可以用一组可预测模式进行描述。

Grady Booch 在他的《软件体系结构手册》[Boo08] 的改进版本中, 提出了以下几种软件系统的体系结构类型, 包括: 人工智能、通信、设备、金融、游戏、工业、法律、医疗、军事、操作系统、运输、实用程序以及许多其他类型。

[257]

13.3 体系结构风格

当建筑师用短语“殖民式中厅”(center hall colonial) 来描述某座房子时, 大多数熟悉美国房子的人能够产生一种整体画面, 即房子看起来是什么样子以及主平面图看起来是什么样子。建筑师使用体系结构风格作为描述手段, 将该房子和其他风格 (例如, A 框架、砖房、鲑鱼角式等) 的房子区分开来。但更重要的是, 体系结构风格也是建筑的样板。必须进一步规定房子的细节, 具体说明它的最终尺寸, 进一步给出定制的特征, 确定建筑材料等。实际上是建筑风格——“殖民式中厅”——指导了建筑师的工作。

基于计算机系统构造的软件也展示了众多体系结构风格中的一种。每种风格描述一种系统类别, 包括: (1) 完成系统需要的某种功能的一组构件 (例如, 数据库、计算模块); (2) 能使构件间实现“通信、合作和协调”的一组连接件; (3) 定义构件如何集成为系统的约束; (4) 语义模型, 能使设计者通过分析系统组成成分的已知属性来理解系统的整体性质 [Bas03]。

体系结构风格就是施加在整个系统设计上的一种变换, 目的是为系统的所有构件建立一个结构。在对已有体系结构再工程 (第 36 章) 时, 体系结构风格的强制采用会导致软件结构的根本性改变, 包括对构件功能的再分配 [Bos00]。

与体系结构风格一样, 体系结构模式也对体系结构设计施加一种变换。然而, 体系结构

关键点 许多不同的体系结构风格可以用于一种特定的类型 (也称为应用领域)。

引述 每位艺术家的思想背后都蕴涵着某种体系结构的模式或者类型。

G. K. Cheserton

提问 什么是体系结构风格?

模式与体系结构风格在许多基本方面存在不同：(1) 模式涉及的范围要小一些，它更多集中在体系结构的某一方面而不是体系结构的整体；(2) 模式在体系结构上施加规则，描述了软件是如何在基础设施层次（例如并发）[Bos00] 上处理某些功能性方面的问题；(3) 体系结构模式（13.3.2 节）倾向于在体系结构环境中处理特定的行为问题（例如，实时应用系统如何处理同步和中断）。模式可以与体系结构风格结合起来建立整个系统结构的外形。

13.3.1 体系结构风格的简单分类

在过去的 60 年中，尽管已经创建了数百万的计算机系统，但绝大多数都可以归为少数的几种体系结构风格之一。

以数据为中心的体系结构。数据存储（如文件或数据库）位于这种体系结构的中心，其他构件会经常访问该数据存储，并对存储中的数据进行更新、增加、删除或者修改。图 13-1 描述了一种典型的以数据为中心的体系结构风格，其中，客户软件访问中心存储库。在某些情况下，数据存储库是被动的，也就是说，客户软件独立于数据的任何变化或其他客户软件的动作而访问数据。该方法的一个变种是将中心存储库变换成“黑板”，当客户感兴趣的数据发生变化时，它将通知客户软件。

以数据为中心的体系结构促进了可集成性（integrability）[Bas03]，也就是说，现有的构件可以被修改，而且新的客户构件可以加入到体系结构中，而无需考虑其他的客户（因为客户构件是独立运作的）。另外，数据可以在客户间通过“黑板”机制传送（即黑板构件负责协调信息在客户间的传递），客户构件独立地执行过程。

数据流体系结构。当输入数据经过一系列计算构件和操作构件的变换形成输出数据时，可以应用这种体系结构。管道-过滤器模式（图 13-2）拥有一组称为过滤器的构件，这些构件通过管道连接，管道将数据从一个构件传送到下一个构件。每个过滤器独立于其上游和下游的构件而工作，过滤器的设计要针对某种形式的数据输入，并且产生某种特定形式的数据输出（到下一个过滤器）。然而，过滤器没有必要了解与之相邻的其他过滤器的工作。

网络资源 基于属性的体系结构风格（ABAS）可用作软件体系结构的构造块。可从 www.sei.cmu.edu/architecture/abas.html 获得相关信息。

引述 设计模式和风格的使用在工程学科中是非常普遍的。
Mary Shaw,
David Garlan

258

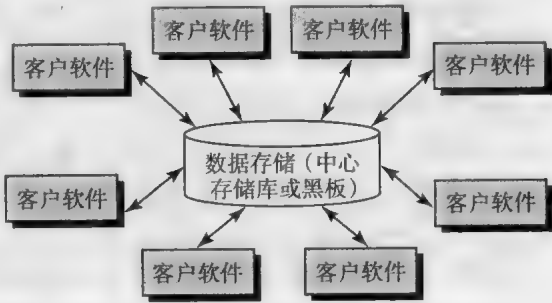


图 13-1 以数据为中心的体系结构

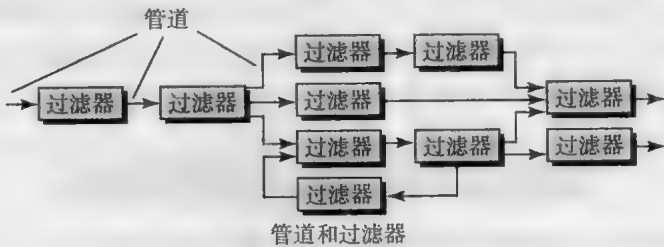


图 13-2 数据流体系结构

如果数据流退化成单线变换，则称为批处理序列（batch sequential）。这种结构接收一批

259 数据，然后应用一系列连续的构件（过滤器）完成变换。

调用和返回体系结构。该体系结构风格能够设计出一个相对易于修改和扩展的程序结构。在此类体系结构中，存在几种子风格 [Bas03]：

- **主程序 / 子程序体系结构。**这种传统的程序结构将功能分解为一个控制层次，其中“主”程序调用一组程序构件，这些程序构件又去调用其他构件。图 13-3 描述了该类型的体系结构。
- **远程过程调用体系结构。**主程序 / 子程序体系结构的构件分布在网络中的多台计算机上。

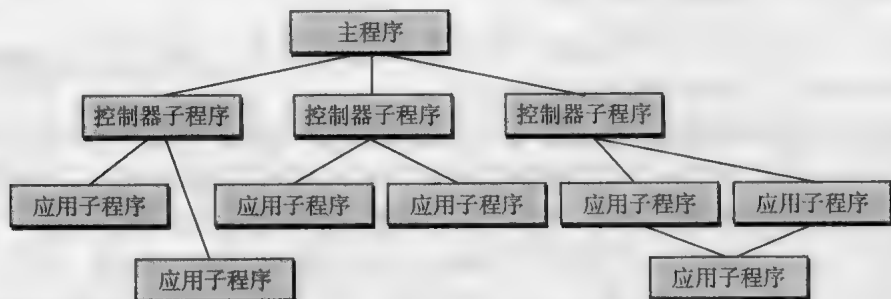


图 13-3 主程序 / 子程序体系结构

260 **面向对象体系结构。**系统的构件封装了数据和必须用于控制该数据的操作，构件间通过信息传递进行通信与合作。

层次体系结构。层次体系结构的基本结构如图 13-4 所示。其中定义了一系列不同的层次，每个层次各自完成操作，这些操作逐渐接近机器的指令集。在外层，构件完成建立用户界面的操作；在内层，构件完成建立操作系统接口的操作；中间层提供各种实用工具服务和应用软件功能。

以上描述的体系结构风格仅仅是可用风格中的一小部分^①。一旦需求工程揭示了待构建系统的特征和约束，就可以选择最适合这些特征和约束的体系结构风格或风格的组合。在很多情况下，会有多种模式是适合的，需要对可选的体系结构风格进行设计和评估。例如，在很多数据库应用中，层次体系结构（适合大多数系统）可以与以数据为中心的体系结构结合起来使用。

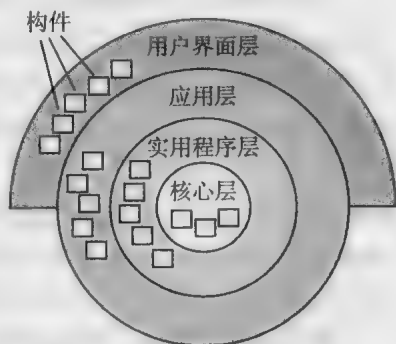


图 13-4 层次体系结构

261 确定一个合适的体系结构风格殊为不易，Buschman [Bus10a] 提出的两种补充思想可以对此提供一些指导。问题帧是指在不考虑专业领域知识或程序实现方案细节的前提下，描述反复出现的问题的特征。领域驱动设计则建议软件设计应该反映出你的应用（第 8 章）试图解决的业务问题的领域以及该领域的逻辑。

SafeHome 选择体系结构风格

[场景] Jamie 的房间，设计建模还在继续
进行。

[人物] Jamie 和 Ed，SafeHome 软件工
程团队的成员。

① 体系结构风格与模式的更详细讨论参见 [Roz11]、[Tay09]、[Bus07]、[Gor06] 或 [Bas03]。

[对话]

Ed (皱着眉): 我们已经使用 UML 对安全功能进行了建模……类、关系等都是其中的基本材料, 所以我觉得面向对象体系结构[⊖]应该是合适的方案。

Jamie: 但是……

Ed: 但是……我对于面向对象体系结构理解起来有些困难, 我比较熟悉调用和返回体系结构——一种传统的过程层次。但是面向对象……我不了解, 它看起来属于无组织的一类。

Jamie (微笑): 无组织?

Ed: 是的……我的意思是说我不能想象出实际的结构, 在设计空间中只有设计类。

Jamie: 哦, 那不对。存在类的层次……想想我们为 FloorPlan 对象设计的层次(聚集)(图 12-3)。面向对象的体系结构是结构与类之间相互连接的组合, 你知道, 类之间的相互连接即相互协作。我们可以通过描述详细的类结构、属性、操作和类之间的消息来体现它。

Ed: 我打算花一个小时制定一个调用和返回体系结构, 然后我再考虑面向对象体系结构。

Jamie: Doug 对此不会有什么问题, 他说我们应该考虑其他方案。顺便说一句, 这两种体系结构彼此结合是绝对没有问题的。

Ed: 好, 我知道了。

一个问题帧是指对同一类问题进行概括以用来解决所遇到的问题, 有五种通常与体系结构风格相关联的基础问题帧: 简单工作片段(工具)、需求行为(以数据为中心)、指令行为(指令处理器)、信息显示(观察者)以及转换(管道和过滤器变换)。

现实世界的问题常常与多个问题帧相关, 相应地, 一种体系结构模型有可能是不同问题帧的组合。例如, 在 WebApp 设计中使用的模型-视图-控制器(MVC)体系结构[⊖]可视为两类问题帧(指令行为和信息显示)的组合。在 MVC 中, 最终用户由浏览器窗口发送指令至指令处理器(控制器), 控制器负责对内容(模型)的访问, 并指示信息生成模型(视图)将其转换至浏览器显示。

262

领域建模会影响到体系结构风格的选择, 领域对象的核心特性尤其如此。表示物理对象的领域对象(如传感器或驱动器)应该与表示逻辑对象的领域对象(如任务调度、工作流)区别开来。物理对象必须严格遵守约束, 如连接限制或消耗资源的使用。逻辑对象可以有一些可被取消或解除的温和的实时行为。层次体系结构对领域驱动设计的支持度最好。[Eva04]

13.3.2 体系结构模式

开发需求模型时将会注意到软件必须解决许多问题, 这些问题很广泛, 跨越了整个应用领域。例如, 对于几乎每一个电子商务应用系统, 其需求模型都会遇到下述问题: 我们如何给各方面的客户提供不同类型的产品, 并且允许这些客户在线购买我们的产品?

需求模型也定义了必须对上述问题进行回答的环境。例如, 面向顾客

引述 也许这是在地下室, 让我们上楼去检查。

M. C. Escher

⊖ 可能有一种争议: SafeHome 体系结构的层次应比指明的层次更高。SafeHome 有许多不同的子系统——住宅监控功能、公司的监控点以及运行在房主 PC 上的子系统。在子系统中, 并行过程(即那些监控传感器)和事件处理非常普遍。在产品工程过程中可以做出该层次上的某些体系结构决策, 但在软件工程中, 体系结构设计可能要考虑这些问题。

⊖ MVC 体系结构在第 17 章详细讨论。

销售高尔夫设备的电子商务和面向媒体和大中型公司销售高价工业设备的电子商务，它们的营运环境完全不同。另外，一系列限制和约束可能会影响到需要解决问题的处理方式。

体系结构模式在特定环境和一系列限制与约束下处理特定的应用问题。模式提出了能够作为体系结构设计基础的体系结构解决方案。

本章前面的部分曾提到过，大多数应用系统都符合特定领域或特定类型，适合于这种类型的风格有一种或者多种。例如，一个应用系统的整体体系结构风格可能是“调用和返回”或者“面向对象”型，但在那种风格中，你会遇到一系列常见问题，这些问题最好是用具体的体系结构模式来处理。第16章将对这些问题中的一部分以及体系结构模式进行详细论述。

13.3.3 组织和求精

由于设计过程经常会留下许多种可供选择的体系结构方案，因此建立一组用于评估所导出的体系结构设计的设计标准是非常重要的。下面的问题[Bas03]有助于更深入地了解体系结构风格：

控制。在体系结构中如何管理控制？是否存在清楚的控制层次？如果存在，构件在控制层次中有什么作用？构件如何在系统中传递控制？构件间如何共享控制？控制的拓扑结构（即控制呈现的几何形状）如何？控制是否同步或者构件操作是否异步？

提问 如何评估导出的体系结构风格？

263

数据。构件间如何进行数据通信？数据流是否连续地传递给系统，或数据对象是否零散地传递给系统？数据传递的模式是什么（数据是从一个构件传递到另一个构件，还是数据被系统中的构件全局共享）？是否存在数据构件（如黑板或中心存储库）？如果存在，它们的作用是什么？功能构件如何和数据构件进行交互？数据构件是被动的还是主动的（数据构件是否主动地和系统中的其他构件进行交互）？系统中的数据和控制如何进行交互？

这些问题有助于设计者对设计质量进行早期评估，也为更详细的体系结构分析奠定了基础。

演化过程模型（第4章）已变得非常流行，这意味着软件体系结构可能需要随着每次产品增量的计划与实施而演变发展。在第12章中我们将此过程描述为重构，即在不改变产品外在行为的情况下对其内部结构进行改进。

13.4 体系结构考虑要素

Buschmann 和 Henny [Bus10b, Bus10c] 提出了几个考虑要素，指导软件工程师在体系结构设计时做出决策。

- **经济性**——许多软件体系结构深受不必要的复杂性所害，它们充斥着不必要的产品特色或无用的需求（如无目的的可重用性）。最好的软件应该是整洁的并依赖抽象化以减少无用的细节。
- **易见性**——设计模型建立后，对于那些随后将验证这些模型的软件工程师而言，体系结构的决策及其依据应该是显而易见的。如果重要的设计和专业领域概念与随后的设计和开发人员没有进行有效沟通，所产生的设计模型往往是晦涩难懂的。
- **隔离性**——不产生隐藏依赖的关注点分离是非常理想的设计思想（第12章），有时我们将此称为隔离性。适当的隔离会产生模块化的设计，但过分的隔离又会导致碎片

提问 开发软件体系结构时应该考虑哪些要素呢？

化和易见性的丧失。诸如领域驱动的设计方法可以协助确定哪些应当分离，哪些应当处理为连贯的单元。

- **对称性**——体系结构的对称性意味着它的属性是均衡一致的，对称的设计更易于理解、领悟和沟通。比如，设想一个客户账户的对象，其生命周期可被软件体系结构直接模式化为同时提供 `Open()` 和 `Close()` 方法。体系结构的对称性可同时包括结构上的对称性和行为上的对称性。
- **应急性**——紧急的、自组织的行为和控制常常是创建可扩展的、经济高效的软件体系结构的关键。比如：许多实时性的软件应用是由事件驱动的，定义系统行为的事件序列和它们的持续时间确定了应急响应的质量，很难预先规划好事件所有可能的序列，相反，系统体系结构设计师应构建一个灵活系统，能够适应这类突发事件的出现。

264

以上这些考虑要素并非独立存在，他们之间既相互作用又相互调节。例如，隔离性可因经济性而加强或减轻，隔离性也可平衡易见性。

软件产品的体系结构描述在实现它的源代码中并非显而易见。相应地，随着源代码的不断修改（如软件维护活动），软件体系结构也将逐渐被侵蚀。对设计者而言，如何对体系结构信息进行适当的抽象是一项挑战。这些抽象可潜在地提高源代码的结构化程度，从而改善可读性和可维护性 [Bro10b]。

SafeHome 评估体系结构决策

[场景] Jamie 的房间，设计建模还在继续进行。

[人物] Jamie 和 Ed，SafeHome 软件工程师团队的成员。

[对话]

Ed: 我完成了安全功能的调用返回体系结构模型。

Jamie: 太好了，你觉得达到了我们的要求吗？

Ed: 它没有任何不必要的功能，看起来很简洁。

Jamie: 易见性如何？

Ed: 还不错，我觉得这个模型用来实现这个产品的安全需求毫无问题。

Jamie: 我相信你已经理解了该体系结构，但你可能不负责该部分的程序开发，我有一点担心隔离性，作为面向对象的设计，这个模型可能还不够模块化。

Ed: 可能是这样，但是当我们创建 SafeHome 的 Web 版的时候，我担心会限

制代码重用的能力。

Jamie: 对称性如何呢？

Ed: 也还可以，我比较难以评估这一块，因为对我而言，在安全功能中仅有的存在对称性的地方就是增加和删除 PIN 信息。

Jamie: 那么当我们在 Web 版中增加远程安全模块时会变得更复杂了。

Ed: 我想是这样。

（考虑到体系结构的异议，他们同时陷入了沉思。）

Jamie: SafeHome 是一个实时系统，所以状态的转换和事件的次序比较难以预见。

Ed: 是的，不过这个系统的应急响应可以用一个有限状态模型来处理。

Jamie: 如何处理呢？

Ed: 该模型可以基于调用返回结构来实现，在很多编程语言中，中断都比较容易处理。

Jamie: 你认为我们有必要对最初考虑的面向对象体系结构进行同样的分析吗？

Ed: 好主意, 一旦开始实施后, 体系结构除安全之外的其他非功能性需求也很重要, 就很难变动了。以确保它们都被通盘考虑了。

265 Jamie: 在这些体系结构之上, 再审视一下 Ed: 对。

13.5 体系结构决策

与系统体系结构相关的决策记录了关键的设计问题以及所选用的体系结构方案背后的原理。一些决策包括软件系统组织、结构元素的选取和它们之间的协作意向所定义的接口, 以及这些元素组合而成的越来越大的子系统 [Kru09]。另外, 还需要进行体系结构模式、应用技术、中间件及编程语言的选择。体系结构决策的成果会影响到系统的非功能特性及其众多的质量属性 [Zim11], 这些成果能够以开发者笔记的形式记录下来。这些笔记记录了关键性的设计决策以及支撑它们的理由, 为新的项目团队成员提供了参考, 也可以作为经验学习的知识库。

通常, 软件体系结构实践的着眼点是呈现和记录不同类别参与者的需要。然而, 定义一种决策视图来贯穿传统的体系结构表现手段中所包含的多种信息视角是有可能的。这种决策视图捕获了体系结构设计决策以及做出决策的依据。

面向服务的体系结构决策 (SOAD)^①建模 [Zim11] 是一种知识管理框架, 它以一种可以指导未来开发活动的方式, 为捕获体系结构决策的依赖提供了支持。

将一种体系结构风格应用于某一特定的应用类型时, 指导模型包含了此体系结构决策所要求的相关知识。该模型基于已完成项目中获取的体系结构信息而建立, 并且此类项目采用了前述的体系结构风格。指导模型记载了设计问题存在的地方、应当做出体系结构决策的地方, 以及从潜在可选方案中作挑选时应当考虑的质量属性。潜在的可选方案 (以及各自的利弊) 是从之前的软件应用中总结出来的, 以辅助体系结构设计师做出最好的决策。

决策模型记录了所需要的体系结构决策、在过往的项目中实际做出的决策以及支持这些决策的理由。指导模型为体系结构决策模型提供了一种可裁剪的步骤, 它允许体系结构设计师删除不相关的议题、扩展重要的议题或是添加新的议题。一个决策模型可以利用多个指导模型, 并在项目完成后向指导模型提供反馈。这种反馈可从项目完成后的经验总结评审中发掘出来。

引述 医生可以文过饰非, 但是建筑师只能建议他的客户牵萝补屋。

Frank Lloyd Wright

13.6 体系结构设计

在体系结构设计开始的时候, 应先建立相应的环境。为达成此目标, 应该定义与软件交互的外部实体 (其他系统、设备、人) 和交互的特性。这些信息一般可以从需求模型中获得。一旦建立了软件的环境模型, 并且描述出所有的外部软件接口, 就可以确定体系结构原型集。

原型是表示系统行为元素的一种抽象 (类似于类)。这个原型集提供了一个抽象集, 如果要使系统结构化, 就必须要对这些原型进行结构化建

提问 什么是原型?

① SOAD 类似于第 16 章中所讨论的体系结构模式的应用, 更多的资料可由此链接中获取: <http://soadecisions.org/soad.htm>。

模,但原型本身并不提供足够的实施细节。因此,设计人员通过定义和细化实施每个原型的软件构件来指定系统的结构。这个过程持续迭代,直到获得一个完善的体系结构。

当软件工程师建立有真实意义的体系结构图时,应先自问并回答一系列问题 [Boo11b]。该图是否能显示系统对输入或事件的响应?哪些部分可以可视化地表达出来,以突出显示风险领域?如何将隐藏的系统设计模式展现给其他开发者?可否以多个视角展现最佳路径以分解系统的特定部分?设计中的各种权衡取舍能否有意义地展现出来?如果一个软件体系结构的图示可以回答以上这些问题,那么对于使用它的软件工程师而言将是很有价值的。

13.6.1 系统环境的表示

在体系结构设计层,软件体系结构设计师用体系结构环境图 (Architectural Context Diagram, ACD) 对软件与其外围实体的交互方式进行建模。图 13-5 给出了体系结构环境图的一般结构。

根据图中所示,与目标系统(为该系统所开发的体系结构设计)交互的系统可以表示为:

- 上级系统——这些系统把目标系统作为某些高层处理方案的一部分。
- 下级系统——这些系统被目标系统使用,并为完成目标系统的功能提供必要的数据和处理。
- 同级系统——这些系统在对等的基础上相互作用(即信息或者由同级系统和目标系统产生,或者被目标系统和同级系统使用)。
- 参与者——通过产生和消耗必要处理所需的信息,实现与目标系统交互的实体(人、设备)。

每个外部实体都通过某一接口(带阴影的小矩形)与目标系统进行通信。

为了说明 ACD 的使用,再来考虑 SafeHome 产品的住宅安全功能。整个 SafeHome 产品的控制器和基于因特网的系统对于安全功能来说都处于上一级,在图 13-6 中它们在上方。监视功能是一个同级系统,并且在以后的产品版本中还要使用住宅安全功能(或被住宅安全功能使用)。房主和控制面板都是参与者,它们既是安全软件所用信息的生产者,又是安全软件所供信息的使用者。最后,传感器为安全软件所使用,并且在图中显示为下一级。

作为体系结构设计的一部分,必须说明图 13-6 中每个接口的细节。目标系统所有的流入和流出数据必须在这个阶段标识出来。

13.6.2 定义原型

原型 (archetype) 是表示核心抽象的类或模式,该抽象对于目标系统体系结构的设计非常关键。通常,即使设计相对复杂的系统,也只需要相对较小的原型集合。目标系统的体系

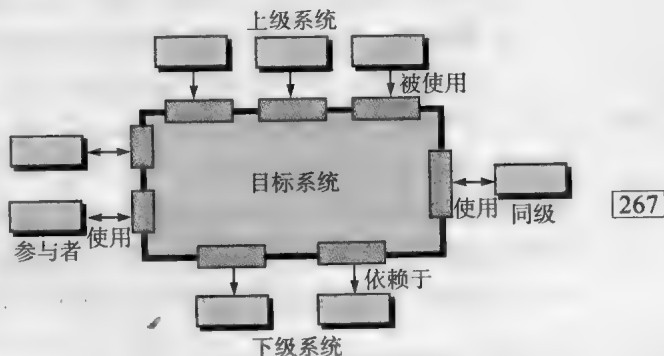


图 13-5 体系结构环境图 [Bos00]

提问 系统之间如何交互?

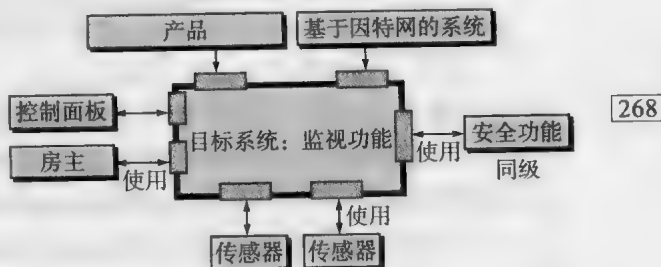


图 13-6 SafeHome 安全功能的体系结构环境图

结构由这些原型组成, 这些原型表示体系结构中稳定的元素, 但可以基于系统行为以多种不同的方式对这些元素进行实例化。

关键点 原型是体系结构设计的抽象构造块。

很多情况下, 可以通过检验作为需求模型一部分的分析类来导出原型。继续关于 SafeHome 住宅安全功能的讨论, 可能会定义下面的原型:

- **结点**。表示住宅安全功能的输入和输出元素的内聚集合, 例如, 结点可能由如下元素构成: (1) 各种传感器; (2) 多种警报 (输出) 指示器。
- **探测器**。对所有为目标系统提供信息的传感设备的抽象。
- **指示器**。表示所有指示警报条件发生的报警机械装置 (例如, 警报汽笛、闪灯、响铃) 的抽象。
- **控制器**。对允许结点发出警报或者撤销警报的机械装置的抽象。如果控制器安装在网络上, 那么它们应该具有相互通信的能力。

269

图 13-7 显示了使用 UML 符号对每一个原型的描述结果。回想那些构成体系结构基础的原型, 它们是抽象的, 随着体系结构设计的进行, 这些抽象必须被进一步求精。例如, 探测器可以被细化为传感器的类层次。

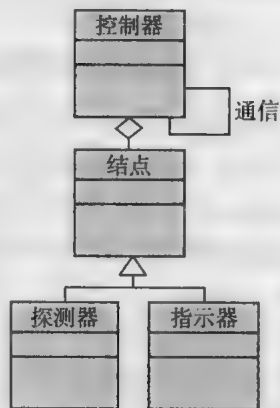


图 13-7 SafeHome 安全功能原型的 UML 关系 [Bos00]

13.6.3 将体系结构细化为构件

在将软件体系结构细化为构件时, 系统的结构就开始显

现了。但是, 如何选择这些构件呢? 为了回答这个问题, 先从需求模型所描述的类开始^①。这些分析类表示软件体系结构中必需处理的应用 (业务) 领域的实体。因此, 应用领域是构件导出和细化的一个源泉。另一个源泉是基础设施域。体系结构必须提供很多基础设施构件, 使应用构件能够运作, 但是这些基础设施构件与应用领域没有业务联系。例如, 内存管理构件、通信构件、数据库构件和任务管理构件经常集成到软件体系结构中。

体系结构环境图 (13.6.1 节) 描述的接口隐含着一个或者多个特定的构件, 这些构件处理经过接口的数据。在某些情况下 (如图形用户界面), 需要设计具有许多构件的、完整的子系统体系结构。

270

继续 SafeHome 住宅安全功能的例子, 可以定义完成下列功能的顶层构件集合:

- **外部通信管理**——协调安全功能与外部实体 (例如, 基于 Internet 的系统与外部报警通知) 的通信。
- **控制面板处理**——管理所有的控制面板功能。
- **探测器管理**——协调对系统所有探测器的访问。
- **警报处理**——审核所有报警条件并执行相应动作。

每一个顶层构件都必须经过反复的迭代细化, 然后在总的 SafeHome 体系结构中进行定位。每个构件都需要定义设计类 (包含相应的属性和操作)。然而, 重要的是, 在进行构件级设计之前, 不要说明所有属性和操作的设计细节 (第 14 章)。

引述 软件系统的结构提供一种“生态环境”, 代码在这个环境里诞生、成长和消亡。一个设计良好的“生态环境”允许软件系统所需要的所有构件成功进化。

R. Pattis

① 如果选择了常规方法 (非面向对象), 那么构件可能从子程序调用层中导出 (图 13-3)。

图 13-8 描述了整体体系结构 (由 UML 构件图表示)。当事务从处理 SafeHome 的 GUI (图形用户界面) 和 Internet 接口的构件进入时, 它们就被外部通信管理获取。该信息由用于选择合适产品功能 (安全功能) 的 SafeHome 执行者构件来管理。控制面板处理构件通过与房主交互来实现安全功能的安装或解除。探测器管理构件定时查询传感器以检测报警条件, 一旦检测到警报, 警报处理构件将产生相应的输出。 [271]

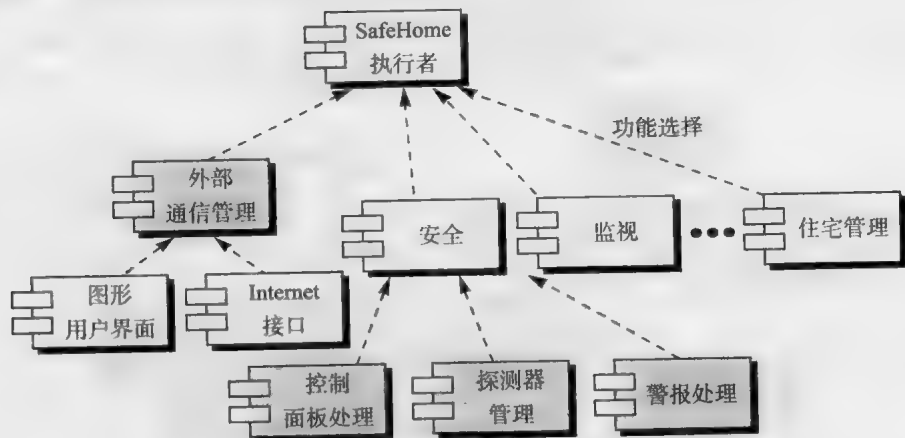


图 13-8 带有顶层构件的 SafeHome 整体体系结构

13.6.4 描述系统实例

到目前为止, 所建模的体系结构设计依然处于比较高的层次。系统环境已经被描述, 问题域中重要抽象的原型已经被定义, 系统的整体结构已经显现, 并且主要的软件构件也都确定了。然而, 更进一步的细化 (回想一下所有的设计都是迭代的) 仍然是必要的。

为了进行体系结构求精, 需要开发体系结构的实际用例。这样做的用意是将体系结构应用到特定问题上, 证明结构和构件都是合理的。

图 13-9 描述的是安全系统 SafeHome 体系结构的一个实例。图 13-8 中显示的构件被进一步细化以显示更多的细节。例如, 探测器管理构件与调度器基础设施构件相互作用, 此基础设施构件实现安全系统中使用的每个传感器对象的定时查询。图 13-8 中显示的所有构件都作了类似的细化。 [272]

软件工具 体系结构设计

[目标] 体系结构设计工具通过描述构件接口、依赖与联系以及交互作用来建立整体软件结构模型。

[机制] 工具采用的机制多种多样。在大多数情况下, 体系结构的设计能力是分析和设计建模自动化工具的一部分功能。

[代表性工具]^①

- Adalon。由 Synthis 公司 (www.synthis.com) 开发, 是一种专用设计工具, 用于设计和构建特定的基于 Web 构件的体系结构。
- ObjectiF。由 microTOOL GmbH (www.

① 这里提到的工具只是此类工具的例子, 并不代表本书支持使用这些工具, 在大多数情况下, 工具名称被各自的开发者注册为商标。

microtool.de/objectiF/en/) 开发，是一个基于 UML 的设计工具，它导出的体系结构（例如 Coldfusion、J2EE 和 Fusebox 等）与基于构件的软件工程（第 14 章）相符。

● Rational Rose。由 Rational (<http://www-01.www-306.ibm.com/software/rational/>) 开发，是基于 UML 的设计工具，它支持体系结构设计中的所有方面。

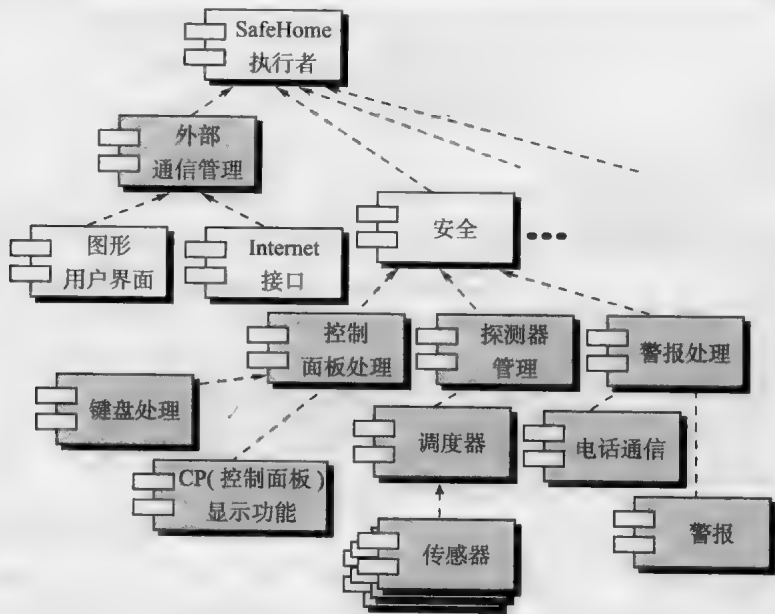


图 13-9 构件细化的安全功能实例

13.6.5 WebApp 的体系结构设计

WebApp[Ⓔ]是典型的使用多层次体系结构来构造的客户端-服务器应用软件，包括用户界面或表现层、一个基于一组业务规则来指导与客户端浏览器进行信息交互的控制器，以及可以包含 WebApp 的业务规则的内容层或模型层。

WebApp 的用户界面是围绕着运行在客户端（通常为个人计算机或移动设备）上的浏览器的特性来设计的。数据层位于服务器。业务规则既可以使用基于服务器的脚本语言（如 PHP）实现，也可以使用基于客户端的脚本语言（如 Javascript）来实现。体系结构设计师应根据安全性和可用性的需求来分配客户端和服务端的功能。

WebApp 的体系结构设计也受客户端所访问的内容结构（线性或非线性的）的影响。WebApp 的体系结构构件（Web 页）被设计为可控的，以传递给系统的其他构件，允许非常灵活的导航结构。媒体以及其他内容资源的物理位置也会对软件工程师确定体系结构产生影响。

[273]

13.6.6 移动 App 的体系结构设计

移动 App[Ⓕ]是典型的使用多层体系结构来构造的系统，包括用户界面层、业务层以及数据层。对于移动 App，你可以选择建立一个基于 Web 的瘦客户端或是一个富客户端。对于瘦客户端，只有用户界面位于移动设备上，业务层和数据层都在服务端。而对于富客户端，

Ⓔ WebApp 将在第 17 章中进行深入的探讨。
Ⓕ 移动 App 设计将在第 18 章进行深入探讨。

所有三层都位于移动设备本身。

每一部移动设备都因它们的物理特性（如屏幕大小、输入设备）、软件（如操作系统、所支持的语言）以及硬件（如内存、网络连接）的不同而有所不同。每一种特性都勾画出了可供选择的体系结构的方向。Meier 和他的同事 [Mei09] 建议了许多考虑要素，这些要素可能影响移动 App 体系结构的设计：（1）将要建立的 Web 客户类型（瘦或富客户端）；（2）所支持的设备种类（如智能手机、平板电脑）；（3）连接程度需求（偶尔的还是持久的）；（4）带宽需求；（5）移动平台的限制；（6）重用和可维护性的程度也是重要的；（7）设备资源的限制（如电池寿命、内存大小、处理器速度）。

13.7 评估候选的体系结构设计

Clements 和他的同事 [Cle03] 在他们关于软件体系结构进化的著作中声称：

说穿了，体系结构就是一个赌注，将注下在系统的成功上。如果你已经提前把赌注放在赢家，而不是等到系统快完成的时候才知道它是否满足需求，这样不是很好吗？如果你准备购买一个系统或者为系统开发付费，难道你不愿意有某种保证，使得系统不偏离正确的道路？如果你自己就是系统架构师，难道你不愿意有一种好方法来验证你的直觉和经验，从而知道所依赖的设计是有基础的，夜里能睡得踏实？

回答这些问题确实有价值。设计会导致多种可选的体系结构，其中每一种可选体系结构都需要进行评估，以确定哪种体系结构最适合要解决的问题。在下面几节中，我们提出两种不同的候选体系结构设计的评估方法。第一种方法使用迭代方法评估设计权衡，第二种方法运用伪定量技术来评估设计质量。

274

软件工程研究所（SEI）开发了一种体系结构权衡分析方法（Architecture Trade-off Analysis Method, ATAM）[Kaz98]，该方法建立了一个迭代的软件体系结构评估过程。下面的设计分析活动是迭代进行的：

1. 收集场景。开发一组用例（第8章和第9章），从用户角度描述系统。
2. 引出需求、约束和环境描述。这些信息作为需求工程的一部分得到确定，并用来确保所有利益相关者的关注点都会被处理。
3. 描述那些已经被选择用于解决场景和需求的体系结构风格或模式。体系结构风格应该使用下面任意一种体系结构视图来描述：
 - 模块视图用于分析带有构件的工作任务以及信息隐蔽的程度。
 - 过程视图用于分析系统性能。
 - 数据流视图用于分析体系结构满足功能需求的程度。
4. 通过单独考虑每个属性来评估质量属性。用于分析的质量属性个数是评审可用时间和质量属性与现存系统相关程度的函数。体系结构设计评估的质量属性包括可靠性、性能、安全性、可维护性、灵活性、可测试性、可移植性、可复用性和互操作性。
5. 针对特定的体系结构风格，确定质量属性对各种体系结构属性的敏感性。这可以通过对体系结构做小的变更并确定某质量属性（如性能）对该变更的敏感性来完成。受体系结构变更影响很大的属性称为敏感点。
6. 使用在第5步进行的敏感性分析来鉴定候选体系结构（在第3步开发的）。SEI 用如下方式描述该方法 [Kaz98]：

一旦确定了体系结构敏感点，寻找这种权衡点就简单了，只需识别出对多个属性敏感的

体系结构元素。例如，客户-服务器体系结构的性能可能对服务器数量是高度敏感的（在一定范围内，增加服务器的数量可使性能提高）……那么，服务器数量就是该体系结构的一个权衡点。

[275]

这6个步骤描述了首次 ATAM 迭代。基于第5步和第6步的结果，某些候选体系结构可能被删除，剩余的一个或多个体系结构可能被修改和进一步细化，然后，再次应用 ATAM 步骤^①。

SafeHome 体系结构评估

[场景] Doug Miller 的办公室，体系结构设计建模正在进行。

[人物] Vinod、Jamie 和 Ed，SafeHome 软件工程团队成员；Doug Miller，软件工程团队经理。

[对话]

Doug：我知道大家为 SafeHome 这个产品设计了两个不同的体系结构，这是一件好事。我的问题是，我们该如何选择最好的体系结构呢？

Ed：我正在设计一个调用和返回风格的体系结构，然后 Jamie 或我将设计一个面向对象的体系结构。

Doug：好的，但是我们如何选择呢？

Jamie：我在高年级时学习了一门计算机科学课程，我记得有很多选取办法。

Vinod：是有一些，但是它们都太理论化了。听着，我认为我们可以自己评估，并使用用例和场景来选择正确的体系结构。

Doug：这不是一回事吗？

Vinod：当谈论有关体系结构评估的时候，它们不是一回事。我们已经有了一个完整的用例集合，因此，我们将每个用例都应

用到这两个体系结构中，查看系统的反应，即在用例环境中构件和连接件是如何工作的。

Ed：这是个好主意！可以确保我们不遗漏任何东西。

Vinod：当然，它还能告诉我们体系结构设计是不是令人费解，系统是不是必须转换到它的分支上来完成工作。

Jamie：场景不就是用例的别名吗？

Vinod：不是的，在这里场景具有不同的意义。

Doug：你们谈论的是质量场景或者变更场景，是吗？

Vinod：是的，我们要做的就是回到利益相关者那里，问问他们 SafeHome 在未来三年可能会有什么变更，如新版本、特征等这类变更。我们创建了一套变更场景，也开发了一套质量场景，这些场景定义了软件体系结构中要看到的属性。

Jamie：我们把它们运用到体系结构中。

Vinod：是的，能更好地处理用例和场景的体系结构就是我们的最终选择。

13.7.1 体系结构描述语言

体系结构描述语言（Architectural Description Language, ADL）提供了一种描述软件体系结构的语义和语法。Hofmann 和他的同事 [Hof01] 建议 ADL 应该使设计者具有分解体系结构构件、将单独构件组合成大的体系结构块以及描述构件之间接口（连接机制）的能力。一

[276]

① 软件体系结构分析方法（SAAM）是 ATAM 的替代方案，值得对体系结构分析感兴趣的读者进行研究。可以从 www.sei.cum.edu/publications/articles/saam-metho-propert-sas.html 网站上下载关于 SAAM 的论文。

且进行了描述,就建立了基于语言技术的体系结构设计,当设计演化时,则更可能建立起有效的体系结构评估方法。

软件工具 体系结构描述语言

下面这些重要的 ADL 总结来自 Rickard Land[Lan02],已得到作者的允许。需要说明的是,前5个ADL是为了研究目的而开发的,它们不是商业产品。

- xArch (<http://www.isr.uci.edu/projects/xarchuci/>) 是一种标准的、可扩展的基于XML的软件体系结构表示方法。
- UniCon ([www.cs.cmu.edu/~ UniCon](http://www.cs.cmu.edu/~UniCon/)) 是“一种体系结构描述语言,旨在帮助设计者用他们发现的很有用的抽象形式定义软件体系结构。”
- Wright ([www.cs.cmu.edu/~ able/wright/](http://www.cs.cmu.edu/~able/wright/)) 是一种包含如下元素的形式化语言:带有端口的构件,带有角色的连

接件,以及将角色连接到端口上的粘合剂。可以使用谓词语言规范体系结构风格,从而允许静态检查,以确定体系结构的一致性和完整性。

- Acme ([www.cs.cmu.edu/~ acme/](http://www.cs.cmu.edu/~acme/)) 是第二代ADL,换句话说,它的目的是确定一种最少共同点ADL。
- UML (www.uml.org/) 包括很多体系结构描述所需要的部分——过程、结点、视图等。对于非正式的描述,UML是相当适用的,因为它有广泛的理解标准。然而,UML没有足够的能力进行体系结构的充分描述。

13.7.2 体系结构评审

体系结构评审是一种特定的技术性评审(第20章),它提供了一种评估方法,该方法可以评估软件体系结构满足系统质量需求(如可扩展性或性能)的能力以及识别任何潜在风险的能力。体系结构评审可以尽早检测到设计问题,具有降低项目成本的潜能。

与需求评审会涉及所有利益相关者的代表不同,体系结构评审往往只涉及软件工程团队成员,并辅以独立的专家。业界最常用的体系结构评审技术有:基于经验的推理^①、原型评估、情境评审(第9章)以及检查单的使用^②。许多体系结构的评审在项目生命周期的早期就开始了,当在基于构件的设计(第14章)中需要新增构件或程序包时也应进行体系结构评审。软件工程师们在进行体系结构评审时,有时会发现体系结构的工作成果中有所缺失或不足,这样会使得评审难以完成[Bab09]。

13.8 经验学习

基于软件的系统是由具有各种各样不同需求和观点的人建立起来的。因此,软件体系结构设计师应该在软件团队成员(及其他利益相关者)间凝聚共识,以便为最终的软件产品达成体系结构愿景[Wri11]。

体系结构设计师在创建体系结构时往往关注系统的非功能性需求的长

网络资源 软件体系结构设计的经验学习可从 http://www.sei.cmu.edu/library/abstracts/news-at-sei/01feature_200707.cfm 获得相关信息。

277

① 基于经验的推理是将新的体系结构与过往已使用的类似的体系结构进行对比。

② 典型的检查单可在此链接里找到: <http://www.opengroup.org/architecture/togaf7-doc/arch/p4/comp/clists/syseng.htm>。

期性影响。高级管理人员在业务战略和战术目标的背景中评估体系结构。项目经理经常受交付日期和预算等短期因素所驱使。软件工程师则常常关注他们自己的技术兴趣和所交付的功能。以上每个群体（以及其他相关人员）应致力于达成共识，以使所选定的软件体系结构明显优于其他可选的方案。

Wright [Wri11] 建议了几种决策分析和解决方案（Decision Analysis and Resolution, DAR）的方法，有助于消除分歧和促成协作。这些方法可以帮助增强团队成员的积极参与度，并提高他们认可最终决策的可能性。DAR 方法帮助团队成员以客观的方式来考虑几种可行的体系结构。三个代表性 DAR 方法如下：

- **原因链法**。一种根源分析^①的技术，团队定义好一种体系结构的目标或效果，然后阐述导致目标实现的相关动作。
- **石川鱼骨法**^②。一种图示技术，列出为达成既定的体系结构目标所需的各种可能的操作或原因。
- **思维导图或蜘蛛图**^③。这种图示方法用来表示围绕着一个中心关键词、约束或需求的词汇、概念、任务或软件工程产品。

网络资源 <http://www.infoq.com/articles/ieee-pattern-based-architecture-reviews> 有基于模式的体系结构评审的讨论。

13.9 基于模式的体系结构评审

正式技术评审（第 20 章）可以应用于软件体系结构，从而为管理系统质量属性、发现错误以及避免不必要的返工提供一种手段。然而，现实情况中短暂的开发周期、紧迫的交付日期、反复变更的需求以及小规模的开发团队往往是常态。针对这些情况，一种轻量的、基于模式的体系结构评审（Pattern-Based Architecture Review, PBAR）流程可能是最佳的选择。

PBAR 是一种用来平衡体系结构模式^④与软件质量属性之间关系的评估方法。PBAR 是涉及所有开发者和其他有兴趣的利益相关者的面对面的审计会议。一位来自外部的体系结构、架构模式、质量属性以及应用领域的专业评审员也应该参加。系统体系结构设计师是首选的主持人。

应该在第一次工作原型或可运行的系统骨架^⑤完成后计划一次 PBAR。PBAR 包含以下的迭代步骤 [Har11]：

1. 遍历相关的用例（第 9 章），以确定并讨论系统最重要的质量属性。
2. 结合需求讨论系统体系结构图。
3. 协助评审人员识别所使用的体系结构模式，并将系统结构与模式结构相匹配。
4. 使用现有文档和过往用例，检查体系结构和质量属性，以确定每一种模式对系统质量

① 更多的信息可在此处获取：<http://www.thinkreliability.com/Root-Cause-Analysis-CM-Basics.aspx>。

② 更多的信息可在此处获取：<http://asq.org/learn-about-quality/cause-analysis/tools/overview/fishbone.html>。

③ 更多的信息可在此处获取：<http://mindmappingsoftwareblog.com/5-best-mind-mapping-programs-for-brainstorming/>。

④ 体系结构模式是针对一个体系结构设计问题以及一组特定的条件和约束的通用方案，模式在第 16 章中详细讨论。

⑤ 可运行的系统骨架包含一个支持业务案例中最高优先级的功能需求以及最有挑战性的质量属性的基线体系结构。

属性的影响。

5. 识别并讨论由设计中使用的体系结构模式所引起的质量问题。

6. 针对会议上出现的问题作一个简短的汇总，并对可运行的系统骨架进行相应的修正。

PBAR 非常适用于小规模敏捷团队，只需要相对较少的额外项目时间和精力。只需要很短的准备及评审时间，PBAR 便能够适应不断变更的需求和较短的建设周期，同时，也有助于团队理解系统体系结构。

13.10 体系结构一致性检查

随着软件的进程由设计进入到构建阶段，软件工程师们必须努力确保实施和演变中的系统与规划的体系结构是一致的。许多情形（如需求冲突、技术困难、交付期限的压力）会造成软件偏离原定的体系结构。如果没有定期对体系结构进行一致性检查，无法控制的偏差便会导致体系结构逐渐被侵蚀并影响系统质量 [Pas10]。

静态体系结构一致性分析（Static Architecture-Conformance Analysis, SACA）可以评估已完成的软件系统是否与它的体系结构模型相符合。系统体系结构建模的形式化方法（如 UML）表现了系统构件的静态组织以及构件间的交互。项目经理常用体系结构模型来分配工作任务和评估实施进程。

网络资源 <http://www.cin.ufpe.br/~fcf3/Arquitetura%20de%20Software/arquitetura/getPDF3.pdf> 有体系结构一致性检查的概览。

279

软件工具 体系结构一致性工具

- Lattix 依赖管理器 (<http://www.lattix.com/>)。这个工具包括一种简单语言，以声明实施中必须遵从的设计规则，检测设计规则中的违例，并按依存结构矩阵可视化地表现它们。
- 源代码查询语言 (<http://www.semmle.com/>)。这个工具可用来自动化软件开发任务，如定义并检查体系结构的约束，使用类 Prolog 语言定义面向对象系统的

继承层次结构的递归查询。

- Reflexion 模型 (http://www.iese.fraunhofer.de/en/competencies/architecture/tools_architecture.html#contentPar_textblockwithpics)。SAVE 工具允许软件工程师建立高级模型以抓取系统体系结构，然后定义这个模型与源代码之间的关系。然后 SAVE 将识别模型和代码之间的缺陷或错误。

13.11 敏捷性与体系结构

在一些敏捷开发的支持者眼里，体系结构设计等同于“大设计前期”，他们认为，这会产生不必要的文档和实现不必要的功能。然而，大多数敏捷开发者确实同意在系统复杂的情况下（即产品有大量的需求、许多利益相关者或广泛的地理分布），专注于软件体系结构是重要的 [Fal10]。基于此，有必要在敏捷过程模型中整合新的体系结构设计。

为了作早期的体系结构决策和避免返工需求，以及避免在选择了错误的体系结构时遭遇质量问题，敏捷开发者应基于一个新近发生的用户故事

网络资源 有关体系结构在敏捷软件过程中的角色可在 <http://msdn.microsoft.com/enus/architecture/ff476940.aspx> 中找到。

280

集来预料体系结构元素[⊖]和结构(第5章)。敏捷团队可以通过建立一个体系结构原型(如一个可运行的系统骨架)并开发清晰的体系结构工作产品,与必要的利益相关者进行沟通,以满足体系结构设计需要。

敏捷开发促使软件体系结构设计师与业务和技术团队在迭代过程中反复地紧密合作,以指导良好的体系结构设计方向。Madison [Mad10] 建议使用包含 Scrum、XP 和顺序项目管理要素[⊖]的混合框架。在这个框架中,前期规划设定了体系结构方向,但迅速进入故事情节串联 [Bro10b]。

在故事情节串联中,体系结构设计师为项目分享体系结构的用户故事,并在规划好“冲刺”(工作单元)后,与产品所有者一起对体系结构故事连同业务用户故事排列优先次序。在“冲刺”期间,体系结构设计师与团队一起工作,以确保演变的软件持续表现出较高的结构质量。如果质量水准高,团队可以独立完成剩余的开发;否则,体系结构设计师在“冲刺”期间加入团队。工作单元结束后,体系结构设计师将会在团队向利益相关者正式展示单元工作成果之前复审工作原型的质量。运作良好的敏捷项目要求每次“冲刺”迭代都应交付工作产品(包括体系结构文档)。对每一次“冲刺”浮现出的工作产品和代码进行复审是一种有用的体系结构评审方式。

职责驱动的体系结构(Responsibility-Driven Architecture, RDA)是一个专注于体系结构决策制定的过程。它指出体系结构决策应当在何时制定、如何制定以及由项目团队中的谁来制定。这种方法强调体系结构设计师是团队的服务型领导,而不是独裁的决策制定者,这与敏捷哲学是一致的。体系结构设计师作为主持者,关注并促进开发团队与团队之外的利益相关者(如业务、安全、基础架构等各类人员)之间的合作。

敏捷团队坚持在新的需求出现时自由地做出变更。体系结构设计师想要确保的是体系结构的重要部分已经经过了斟酌,并且开发者也征求了利益相关者的意见。两方的意见可以通过运用一种称为进展的签署的做法来得到满足,即在每一次新的原型完成后,相关的产品应该记录在文档中并获得批准 [Bla10]。

运用与敏捷思想一致的过程可给监管方和审计部门提供可验证的签署,而且不会妨碍敏捷团队做出所需的决策。在项目结束时,团队会有一整套工作产品,体系结构也会在演变过程中获得质量评审。

281

13.12 小结

软件体系结构提供了待构建系统的整体视图,它描述软件构件的结构和组织、构件的属性以及构件之间的连接。软件构件包括程序模块和程序操作的各种数据表示。因此,数据设计是软件体系结构设计的一个组成部分。体系结构注重于早期的设计决策,并提供考虑多个可选系统结构优点的机制。

软件工程师可以使用多种不同的体系结构风格和模式,在给定的体系结构类型中可以使用这些风格和模式。每种风格描述了一种系统类别,它包含:一组完成系统所需功能的构件;一组使构件间通信、协调及合作的连接件;定义如何集成构件以构成系统的约束条件;使设计者能够理解系统整体特性的语义模型。

⊖ 卓越的关于体系结构敏捷性的讨论可在 [Bro10a] 中找到。

⊖ Scrum 和 XP 是敏捷过程模型,在第5章讨论。

一般来说,体系结构设计需要4个不同步骤来完成。第一步,系统必须表示在相应的环境中,也就是说,设计人员应该定义与软件交互的外部实体及其交互性质。一旦环境得到说明,设计人员应该确定一系列的顶层抽象,称之为原型,该原型可以表示系统行为或者功能的关键元素。定义抽象之后,设计开始逐渐被实现。在支持构件的体系结构环境中识别和描述这些构件。最后,开发体系结构的特定实例,在真实世界中验证设计。

通过运用一种混合的体系结构设计框架,可以使体系结构与敏捷方法并存。这种框架使用了现有的由流行的敏捷方法衍生而来的技术。一旦体系结构确立,便可以评估它是否与业务目标、软件需求以及质量属性相一致。

习题与思考题

- 13.1 用一个房屋或建筑物的结构作比喻,与软件体系结构作对照分析。经典建筑与软件体系结构的原则有什么相似之处?又有何区别?
- 13.2 举出2~3个例子,说明13.3.1节中提到的每一种体系结构风格的应用。
- 13.3 13.3.1节中提到的一些体系结构风格具有层次性,而另外一些则没有。列出每种类型。没有层次的体系结构风格如何实现?
- 13.4 在软件体系结构讨论中,经常会遇到体系结构风格、体系结构模式及框架(本书中没有讨论)等术语。研究并描述这些术语之间的不同。
- 13.5 选择一个你熟悉的应用软件,回答13.3.3节中对于控制与数据提出的每一个问题。
- 13.6 研究ATAM(使用[Kaz98])并对13.7.1节提出的6个步骤进行详细讨论。
- 13.7 如果还没有完成问题9.5,请先完成它。使用本章描述的设计方法开发PHTRS的软件体系结构。
- 13.8 使用13.1.4节中的体系结构决策模板为问题13.7中的一个PHTRS体系结构决策撰写文档。
- 13.9 选取一个你熟悉的移动App,使用13.4节中的体系结构要素(经济性、易见性、隔离性、对称性、应急性)来对其进行评估。
- 13.10 列出问题13.7中你完成的PHTRS体系结构的优点及不足之处。
- 13.11 为问题13.7中你完成的PHTRS体系结构创建一个依赖结构矩阵[⊖]。
- 13.12 从第5章中选取一种敏捷过程模型,并标识出它所包含的体系结构设计活动。

282

扩展阅读与信息资源

在过去的10年中,已经有很多关于软件体系结构的文献。Varma(《Software Architecture: A Case Based Approach》, Pearson, 2013)通过一系列的案例研究来展现体系结构。Bass和他的同事(《Software Architecture in Practice》, 3rd ed., Addison-Wesley, 2012), Gorton(《Essential Software Architecture》, 2nd ed., Springer, 2011)、Rozanski和Woods(《Software Systems Architecture》, 2nd ed., Addison-Wesley, 2011)、Eeles和Cripps(《The Process of Software Architecting》, Addison-Wesley, 2009)、Taylor和他的同事(《Software Architecture》, Wiley, 2009)、Reekie和McAdam(《A Software Architecture Primer》, 2nd ed., Angophora Press, 2006)以及Albin(《The Art of Software Architecture》, Wiley, 2003)的书中,对智能领域挑战性的话题给出了有意义的介绍。

Buschman和他的同事(《Pattern-Oriented Software Architecture》, Wiley, 2007)以及Kuchana(《Software Architecture Design Patterns in Java》, Auerbach, 2004)讨论了体系结构设计的面向模式的方面。Knoernschilf(《Java Application Architecture:Modularity Patterns with Examples Using OSGi》,

⊖ 可以维基百科为起点获取更多关于DSM的资讯:http://en.wikipedia.org/wiki/Design_structure_matrix。

Prentice Hall, 2012)、Rozanski 和 Woods (《Software Systems Architecture》, 2nd ed., Addison-Wesley, 2011)、Henderikson (《12 Essential Skills for Software Architects》, Addison-Wesley, 2011)、Clements 和他的同事 (《Documenting Software Architecture: View and Beyond》, 2nd ed., Addison-Wesley, 2010)、Microsoft (《Microsoft Application Guide》, Microsoft Press, 2nd ed., 2009)、Fowler (《Patterns of Enterprise Application Architecture》, Addison-Wesley, 2003)、Bosch[Bos00] 以及 Hofmeister 和他的同事 [Hof00] 提供了软件体系结构的更深入的方法。

Hennesey 和 Patterson (《Computer Architecture》, 5th ed., Morgan-Kaufmann, 2011) 对软件体系结构的设计问题作了清楚的定量描述。Clements 和他的同事 (《Evaluating Software Architecture》, Addison-Wesley, 2002) 对软件体系结构候选方案评估以及给定问题域的最优软件体系结构选取的相关问题进行了研究。

有关体系结构具体实现的书目陈述了特定开发环境和技术中的体系结构设计。Erl (《SOA Design Patterns》, Prentice-Hall, 2009) 以及 Marks 和 Bell (《Service-Oriented Architecture》, Wiley, 2006) 讨论了将业务及计算资源与客户定义需求联系起来的设计方法。Bambilla 等人 (《Model-Driven Software Engineering in Practice》, Morgan Claypool, 2012) 以及 Stahl 和他的同事 (《Model-Driven Software Development》, Wiley, 2006) 讨论了在具体领域建模方法环境中的体系结构。Radaideh 和 Al-ameed (《Architecture of Reliable Web Applications Software》, IGI Global, 2007) 研究了适用于 WebApp 的体系结构。Esposito (《Architecting Mobile Solutions for the Enterprise》, Microsoft Press, 2012) 讨论了移动 App 体系结构。Clements 和 Northrop (《Software Product Lines: Practices and Patterns》, Addison-Wesley, 2001) 论述了支持软件产品线的体系结构。Shanley (《Protected Mode Software Architecture》, Addison-Wesley, 1996) 给出了设计基于 PC 的实时操作系统、多任务操作系统或者设备驱动程序的体系结构设计指导原则。

当前的软件体系结构研究每年都会记录在由 ACM 和其他计算机组织发起的软件体系结构国际研讨会会议录 (Proceedings of the International Workshop on Software Architecture) 和软件工程国际会议会议录 (Proceedings of the International Conference on Software Engineering) 中。

关于体系结构设计的大量的信息资源可以在网上获得。在 SEPA 网站 www.mhhe.com/pressman 上可以找到和体系结构设计相关的最新参考文献。

构件级设计

要点浏览

概念：完整的软件构件是在体系结构设计过程中定义的。但是没有在接近代码的抽象级上表示内部数据结构和每个构件的处理细节。构件级设计定义了数据结构、算法、接口特征和分配给每个软件构件的通信机制。

人员：软件工程师完成构件级设计。

重要性：必须能够在构造软件之前就确定该软件是否可以工作。为了保证设计的正确性，以及与早期设计表示（即数据、体系结构和接口设计）的一致性，构件级设计需要以一种可以评审设计细节的方式来表示软件。它提供了一种评估数据结构、接口和算法是否能够工作的方法。

步骤：数据、体系结构和接口的设计表示构成了构件级设计的基础。每个构件的

类定义或者处理说明都转化为一种详细设计，该设计采用图形或基于文本的形式来详细说明内部的数据结构、局部接口细节和处理逻辑。设计符号包括 UML 图和一些辅助表示，通过使用一系列结构化编程结构来说明过程的设计。通常的做法是采用现有可复用软件构件，而不是开发新的构件。

工作产品：每个构件的设计都以基于图形的表格或文本方式表示，这是构件级设计阶段产生的主要工作产品。

质量保证措施：采用设计评审机制。对设计执行检查以确定数据结构、接口、处理顺序和逻辑条件等是否都正确，并且给出早期设计中与构件相关的数据或控制变换。

体系结构设计第一次迭代完成之后，就应该开始构件级设计。在这个阶段，全部数据和软件的程序结构都已经建立起来。其目的是把设计模型转化为运行软件。但是现有设计模型的抽象层次相对较高，而可运行程序的抽象层次相对较低。这种转化具有挑战性，因为可能会在软件过程后期引入难以发现和改正的微小错误。Edsger Dijkstra（帮助我们理解软件设计技术的主要贡献者）在其著作 [Dij72] 中写道：

软件似乎不同于很多其他产品，对那些产品而言，一条规则是：更高的质量意味着更高的价格。那些想要真正可靠软件的人发现他们必须找到某种方法来避免开始时的大多数错误，结果，程序设计过程的成本更低……高效率的程序员……不应该将他们的时间浪费在调试上——在开始时就不应该引入错误。

尽管这段话是在很多年前说的，但现在仍然适用。当设计模型被转化为源代码时，必须遵循一系列设计原则，以保证不仅能够完成转化任务，而且不在开始时就引入错误。

关键概念

内聚性

构件

适应性

分类

组合

合格性

WebApp

基于构件的开发

内容设计

耦合

依赖倒置原则

设计重用

设计准则

领域工程

14.1 什么是构件

通常来讲,构件是计算机软件中的一个模块化的构造块。再正式一点,OMG 统一建模语言规范 [OMG03a] 是这样定义构件的:系统中模块化的、可部署的和可替换的部件,该部件封装了实现并对外提供一组接口。

正如第 13 章的讨论,构件存在于软件体系结构中,因而构件在完成所建系统的需求和目标的过程中起着重要作用。由于构件驻留于软件体系结构的内部,因此它们必须与其他的构件和存在于软件边界以外的实体(如其他系统、设备和人员)进行通信和合作。

对于术语构件的实际意义,软件工程师可能有不同的见解。在接下来的几节中,我们将了解关于“什么是构件以及在设计建模中如何使用构件”的三种重要观点。

关键概念

接口分离原则
Liskov 替换原则
面向对象观点
开闭原则
过程相关
传统构件
传统观点

引述 细节不仅
是细节,它们构
成设计。

Charles Eames

14.1.1 面向对象的观点

在面向对象软件工程环境中,构件包括一个协作类集合^①。构件中的每个类都得到详细阐述,包括所有属性和与其实现相关的操作。作为细节设计的一部分,必须定义所有与其他设计类相互通信协作的接口。为此,设计师需要从分析模型开始,详细描述分析类(对于构件而言该类与问题域相关)和基础类(对于构件而言该类为问题域提供了支持性服务)。

关键点 以面向
对象的观点来看,
构件是协作类的
集合。

为了说明设计细化过程,考虑为一个高级影印中心构造软件。软件的目的是收集前台的客户需求,对印刷业务进行定价,然后把印刷任务交给自动生产设备。在需求工程中得到了一个名为 PrintJob 的分析类。分析过程中定义的属性和操作在图 14-1 的上方给出了注释。在体系结构设计中,PrintJob 被定义为软件体系结构的一个构件,用简化的 UML 符号表示的该构件显示在图 14-1 中部靠右的位置^②。需要注意的是,PrintJob 有两个接口:computeJob 和 initiateJob。computeJob 具有对任务进行定价的功能,initiateJob 能够把任务传给生产设备。这两个接口在图下方的左边给出(即所谓的棒棒糖式符号)。

构件级设计将由此开始。必须对 PrintJob 构件的细节进行细化,以提供指导实现的充分信息。通过不断补充作为构件 PrintJob 的类的全部属性和操作,来逐步细化最初的分析类。正如图 14-1 右下部分的描述,细化后的设计类 PrintJob 包含更多的属性信息和构件实现所需要的更广泛的操作描述。computeJob 和 initiateJob 接口隐含着与其他构件(图中没有显示出来)的通信和协作。例如,computePageCost() 操作(computeJob 接口的组成部分)可能与包含任务定价信息的 PricingTable 构件进行协作。checkPriority() 操作(initiateJob 接口的组成部分)可能与 JobQueue 构件进行协作,用来判断当前等待生产的任务类型和优先级。

建议 请回想一
下,分析建模和
设计建模都是迭
代动作。细化原
分析类可能需要
额外的分析步骤,
表示细化设计类
的设计建模步骤
紧随其后(构件
的细节)。

对于体系结构设计组成部分的每个构件都要实施细化。细化一旦完成,要对每个属性、每个操作和每个接口进行更进一步的细化。对适合每个属性的数据结构

① 在某些情况下,构件可以包含一个简单的类。

② 不熟悉 UML 表示法的读者可以参考附录 1。

必须予以详细说明。另外还要说明实现与操作相关的处理逻辑的算法细节，在这一章的后半部分将要对这种过程设计活动进行讨论。最后是实现接口所需机制的设计。对于面向对象软件，还包含对实现系统内部对象间消息通信机制的描述。

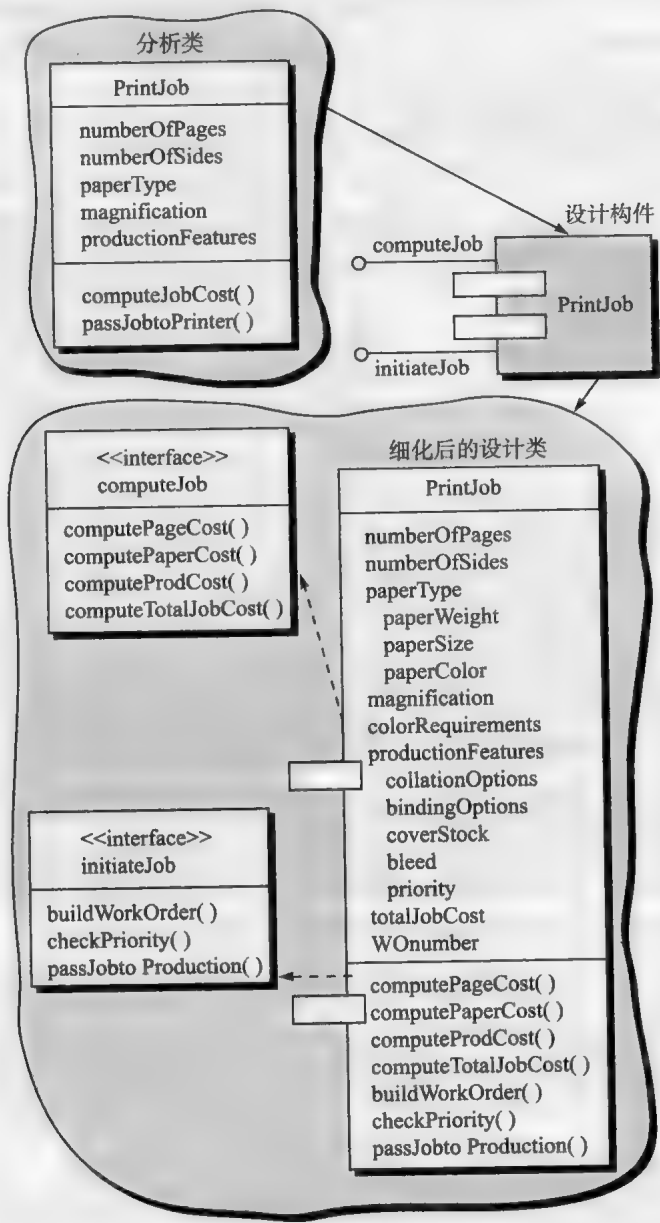


图 14-1 设计构件的细化

14.1.2 传统的观点

在传统软件工程环境中，一个构件就是程序的一个功能要素，程序由处理逻辑及实现处理逻辑所需的内部数据结构以及能够保证构件被调用和实现数据传递的接口构成。传统构件也称为模块，作为软件体系结构的一部分，它扮演如下三个重要角色之一：（1）控制构件，协调问题域中所有其他构件的调用；（2）问题域构件，完成客户需要的全部功能或部分功能；（3）基础设施构件，负责完成问题域中所需的支持处理的功能。

288

与面向对象的构件类似，传统的软件构件也来自于分析模型。不同的是在这种情况下，是以分析模型中的构件细化作为导出构件的基础。构件层次结构上的每个构件都被映射为某一层级上的模块（13.6 节）。一般来讲，控制构件（模块）位于层次结构（体系结构）顶层附近，而问题域构件则倾向位于层次结构的底层。为了获得有效的模块化，在构件细化的过程中采用了功能独立性的设计概念（第 12 章）。

为了说明传统构件的细化过程，我们再来考虑为一个高级影印中心构造的软件。一个分层的体系结构的导出如图 14-2 所示。图中每个方框都表示一个软件构件。带阴影的方框在功能上相当于 14.1.1 节讨论的为 PrintJob 类定义的操作。然而，在这种情况下，每个操作都被表示为如图 14-2 所示的能够被调用的单独模块。其他模块用来控制处理过程，也就是前面提到的控制构件。

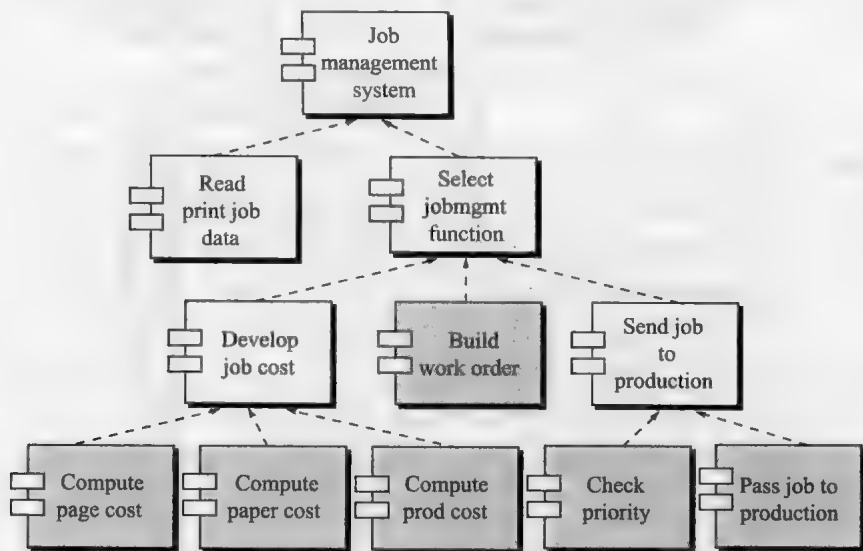


图 14-2 一个传统系统的结构图

在构件级设计中，图 14-2 中的每个模块都要被细化。需要明确定义模块的接口，即每个经过接口的数据或控制对象都需要明确加以说明。还需要定义模块内部使用的数据结构。采用第 12 章讨论的逐步求精方法设计完成模块中相关功能的算法。有时候需要用状态图表示模块行为。

289

为了说明这个过程，考虑 ComputePageCost 模块。该模块的目的在于根据用户提供的规格说明来计算每页的印刷成本。为了实现该功能需要以下数据：文档的页数、文档的印刷份数、单面或者双面印刷、颜色、纸张大小。这些数据通过该模块的接口传递给 ComputePageCost。ComputePageCost 根据任务量和复杂度，使用这些数据来决定一页的成本——这是一个通过接口将所有数据传递给模块的功能。每一页的成本与任务量成反比，与任务的复杂度成正比。

图 14-3 给出了使用改进的 UML 建模符号描述的构件级设计。其中 ComputePageCost 模块通过调用 getJobData 模块（它允许所有相关数据都传递给该构件）和数据库接口 accessCostDB（它能够使该模块访问存放所

引述 可工作的复杂系统都是由可工作的简单系统演化来的。

John Gall

建议 随着每个软件构件设计的逐步细化，设计焦点转向特定数据结构的设计和操作系统结构的过程设计。但是，不要忘了体系结构，它必须容纳构件或服务于多数构件的全局数据结构。

有印刷成本的数据库)来访问数据。接着,对 ComputePageCost 模块进一步细化,给出算法和接口的细节描述(图 14-3)。其中,算法的细节可以由图中显示的伪代码或者 UML 活动图来表示。接口被表示为一组输入和输出的数据对象或者数据项的集合。设计细化的过程一直进行下去,直到能够提供指导构件构造的足够细节为止。

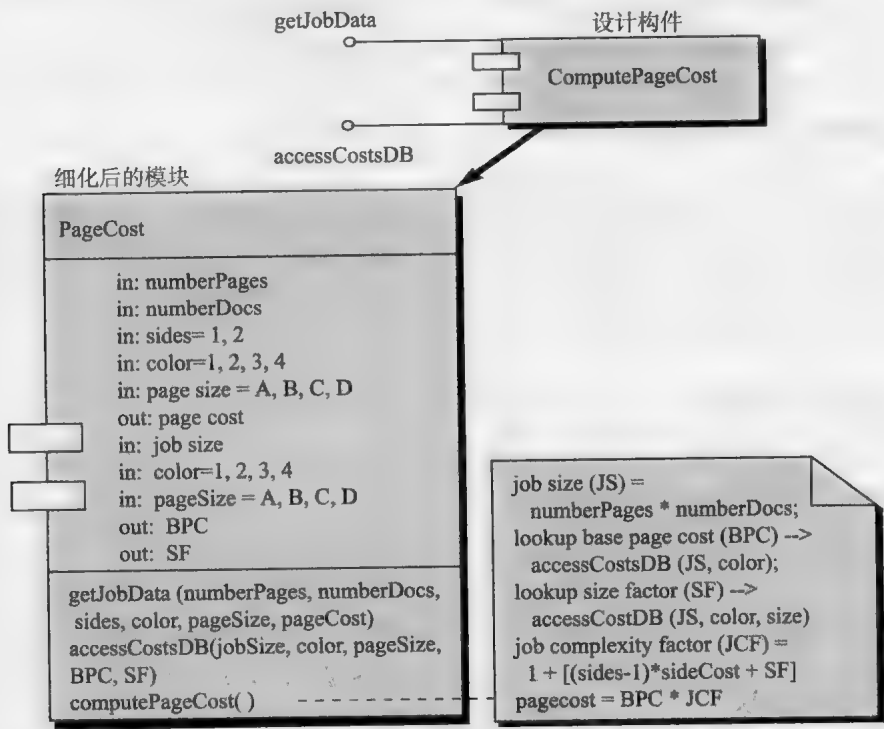


图 14-3 ComputePageCost 的构件级设计

14.1.3 过程相关的观点

290

14.1.1 节和 14.1.2 节提到的关于构件级设计的面向对象观点和传统观点,都假定从头开始设计构件。也就是说,设计者必须根据从需求模型中导出的规格说明创建新构件。当然,还存在另外一种方法。

在过去的 30 年间,软件工程已经开始强调使用已有构件或设计模式来构造系统的必要性。实际上,软件工程师在设计过程中可以使用已经过验证的设计或代码级构件目录。当软件体系结构设计完后,就可以从目录中选出构件或者设计模式,并用于组装体系结构。由于这些构件是根据复用思想来创建的,所以其接口的完整描述、要实现的功能和需要的通信与协作等对于设计者来说都是可以得到的。在 14.6 节将讨论基于构件的软件工程(Component-Based Software Engineering, CBSE)的某些重要方面。

信息栏 基于构件的标准和框架

基于构件的软件工程(CBSE)成功或者失败的重要因素之一就是基于构件标准(有时称为中间件)的可用性。中间件是一组基础构件的集合,这些基础构件使得问

题域构件与其他构件通过网络进行通信,或者在一个复杂的系统中通信。对于想用基于构件的软件工程方法进行开发的软件工程师来讲,他们可以使用如下的标准:

- OMG CORBA——<http://www.corba.org/>。
 - Microsoft COM——<http://www.microsoft.com/com/default.msp>。
 - Microsoft .NET——<http://msdn.microsoft.com/en-us/netframework/default.aspx>。
- Sun JavaBeans——<http://www.oracle.com/technetwork/java/javaee/ejb/index.html>。

上面提到的网站上提供了大量的教材、白皮书、工具和关于这些重要中间件标准的大量资源。

14.2 设计基于类的构件

正如前面提到的，构件级设计利用了需求模型（第 9 ~ 11 章）开发的信息和体系结构模型（第 13 章）表示的信息。选择面向对象软件工程方法之后，构件级设计主要关注需求模型中问题域特定类的细化和基础类的定义和细化。这些类的属性、操作和接口的详细描述是开始构造活动之前所需的设计细节。

291

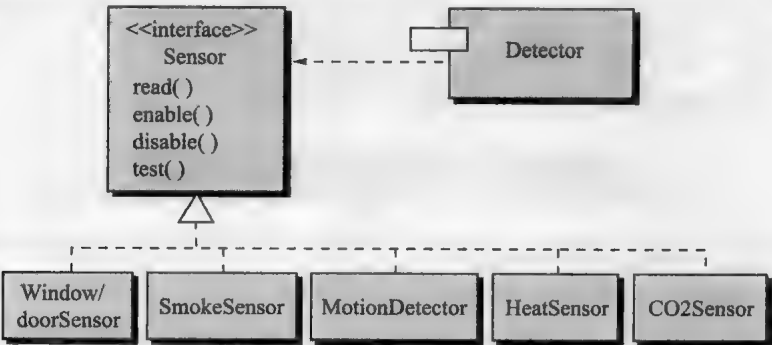
14.2.1 基本设计原则

有四种适用于构件级设计的基本设计原则，这些原则在使用面向对象软件工程方法时被广泛采用。使用这些原则的根本动机在于，使得产生的设计在发生变更时能够适应变更并且减少副作用的传播。设计者以这些原则为指导进行软件构件的开发。

关闭原则（The Open-Closed Principle, OCP）。模块（构件）应该对外延具有开放性，对修改具有封闭性 [Mar00]。这段话似乎有些自相矛盾，但是它却体现出优秀的构件级设计应该具有的最重要特征之一。简单地说，设计者应该采用一种无需对构件自身内部（代码或者内部逻辑）做修改就可以进行扩展（在构件所确定的功能域内）的方式来说明构件。为了达到这个目的，设计者需要进行抽象，在那些可能需要扩展的功能与设计类本身之间起到缓冲区的作用。

例如，假设 SafeHome 的安全功能使用了对各种类型安全传感器进行状态检查的 Detector 类。随着时间的推移，安全传感器的类型和数量将会不断增长。如果内部处理逻辑采用一系列 if-then-else 的结构来实现，其中每个这样的结构都负责一个不同的传感器类型，那么对于新增加的传感器类型，就需要增加额外的内部处理逻辑（依然是另外的 if-then-else 结构），而这显然违背 OCP 原则。

图 14-4 中给出了一种遵循 OCP 原则实现 Detector 类的方法。对于各种不同的传感器，Sensor 接口都向 Detector 构件呈现一致的视图。如果要添加新类型的传感器，那么对 Detector 类（构件）无需进行任何改变。这个设计遵守了 OCP 原则。



292

图 14-4 遵循 OCP 原则

SafeHome OCP 的应用

[场景] Vinod 的工作间。

[人物] Vinod 和 Shakira, SafeHome 软件工程团队成员。

[对话]

Vinod : 我刚刚接到 Doug (团队经理) 的一个电话, 他说市场营销人员想增加一个新的传感器。

Shakira (假笑): 哎呀, 别再加了。

Vinod : 是啊……你永远不可能相信这些家伙都提出了什么。

Shakira : 确实令我很吃惊。

Vinod (大笑): 他们称之为小狗焦虑传感器 (doggie angst sensor)。

Shakira : 那是什么装置?

Vinod : 这个装置是为了那些想把宠物留在彼此相邻很近的公寓、门廊或房子里的人设计的。狗叫致使邻里生气和抱怨, 有了这种传感器, 如果狗的叫声超过一定时间 (比如一分钟), 传感器就会向主人的手机发送特殊的报警信号。

Shakira : 你在开玩笑吗?

Vinod : 不是, Doug 想知道在安全功能中加入这个功能需要多长时间。

Shakira (想了想): 不用多长时间……瞧 (她给 Vinod 看图 14-4), 我们分离出在 sensor 接口背后的实际的传感器类。只要我们有小狗传感器的规格说明, 那么把它加入其中就是一件简单的事情了。我们要做的就是为其创建一个合适的构件……哦, 类。根本不用改变 Detector 构件。

Vinod : 好的, 我将告诉 Doug 这不是什么大问题。

Shakira : 告诉 Doug, 直到下一个版本发布之前, 我们都要集中精力完成小狗焦虑传感器的事情。

Vinod : 这不是件坏事, 而如果他想让你做, 你可以马上实现吗?

Shakira : 是啊, 我们的接口设计使得我可以毫无困难地完成它。

Vinod (想了想): 你听说过开闭原则吗?

Shakira (耸了耸肩膀): 没有。

Vinod (微笑): 不成问题。

Liskov 替换原则 (Liskov Substitution Principle, LSP)。子类可以替换它们的基类 [Mar00]。最早提出该设计原则的 Barbara Liskov [Lis88] 建议, 将从基类导出的类传递给构件时, 使用基类的构件应该仍然能够正确完成其功能。LSP 原则要求源自基类的任何子类必须遵守基类与使用该基类的构件之间的隐含约定。在这里的讨论中, “约定”既是前置条件——构件使用基类前必须为真; 又是后置条件——构件使用基类后必须为真。当设计者创建了导出类, 则这些子类必须遵守前置条件和后置条件。

依赖倒置原则 (Dependency Inversion Principle, DIP)。依赖于抽象, 而非具体实现 [Mar00]。正如我们在 OCP 中讨论的那样, 抽象可以比较容易地对设计进行扩展, 又不会导致大量的混乱。构件依赖的其他具体构件 (不是依赖抽象类, 如接口) 越多, 扩展起来就越困难。

接口分离原则 (Interface Segregation Principle, ISP)。多个客户专用接口比一个通用接口要好 [Mar00]。多个客户构件使用一个服务器类提供的操作的实例有很多。ISP 原则建议设计者应该为每个主要的客户类型都设计一个特定的接口。只有那些与特定客户类型相关的操作才应该出现在该客户的接口说明中。如果多个客户要求相同的操作, 则这些操作应该在每个特定的接口中都加以说明。

例如, 假设 FloorPlan 类用在 SafeHome 的安全和监视功能中 (第 10 章)。对于安全功

建议 如果你省去设计并且破解出代码, 记住代码只是最终的“具体实现”, 你违背了 DIP 原则。

能, FloorPlan 只在配置活动中使用, 并且使用 placeDevice()、showDevice()、groupDevice()、removeDevice() 等操作实现在建筑平面图中放置、显示、分组和删除传感器。SafeHome 监视功能除了需要这四个有关安全的操作之外, 还需要特殊的操作 showFOV()、showDeviceID() 来管理摄像机。因此, ISP 建议为来自 SafeHome 功能的两个客户端构件定义特殊的接口。安全接口应该只包括 placeDevice()、showDevice()、groupDevice()、removeDevice() 四种操作。监视接口应该包括 placeDevice()、showDevice()、groupDevice()、removeDevice()、showFOV()、showDeviceID() 六种操作。

关键点 设计可复用的构件不仅需要优秀的技术设计, 而且需要高效的配置控制机制(第29章)。

尽管构件级设计原则提供了有益的指导, 但构件自身不能够独立存在。在很多情况下, 单独的构件或者类被组织到子系统或包中。于是我们很自然地就会问这个包会有怎样的活动。在设计过程中如何正确组织这些构件? Martin 在 [Mar00] 中给出了在构件级设计中可以应用的另外一些打包原则, 这些原则如下。

发布复用等价性原则 (Release Reuse Equivalency Principle, REP)。复用的粒度就是发布的粒度 [Mar00]。当设计类或构件用以复用时, 在可复用实体的开发者和使用者之间就建立了一种隐含的约定关系。开发者承诺建立一个发布控制系统, 用来支持和维护实体的各种老版本, 同时用户逐渐地将其升级到最新版本。明智的方法是将可复用的类分组打包成能够管理和控制的包并作为一个更新的版本, 而不是对每个类分别进行升级。

共同封装原则 (Common Closure Principle, CCP)。一同变更的类应该合在一起 [Mar00]。类应该根据其内聚性进行打包。也就是说, 当类被打包成设计的一部分时, 它们应该处理相同的功能或者行为域。当某个域的一些特征必须变更时, 只有相应包中的类才有可能需要修改。这样可以进行更加有效的变更控制和发布管理。

294

共同复用原则 (Common Reuse Principle, CRP)。不能一起复用的类不能被分到一组 [Mar00]。当包中的一个或者多个类变更时, 包的发布版本号也会发生变更。所有那些依赖于已经发生变更的包的类或者包, 都必须升级到最新版本, 并且都需要进行测试以保证新发布的版本能够无故障运转。如果类没有根据内聚性进行分组, 那么这个包中与其他类无关联的类有可能会发生变更, 而这往往会导致进行没有必要的集成和测试。因此, 只有那些一起被复用的类才应该包含在一个包中。

14.2.2 构件级设计指导方针

除了 14.2.1 节中讨论的原则之外, 在构件级设计的进程中还可以使用一系列实用的设计指导方针。这些指导方针可以应用于构件、构件的接口, 以及对于最终设计有着重要影响的依赖和继承特征等方面。Ambler[Amb02b] 给出了如下的指导方针。

构件。对那些已经被确定为体系结构模型一部分的构件应该建立命名约定, 并对其做进一步的精细化处理, 使其成为构件级模型的一部分。体系结构构件的名字来源于问题域, 并且对于考虑体系结构模型的所有利益相关者来说都是有意义的。例如, 无论技术背景如何, FloorPlan 这个类的名称对于任何读到它的人来说都是有意义的。另一方面, 基础构件或者细化后的构件级类应该以能够反映其实现意义的名称来命名。例如, 对一个作为 FloorPlan 实现一部分的链表进

提问 命名构件时需要考虑些什么?

行管理时, 操作 `manageList()` 是一个合适的名称, 因为即使是非技术人员也不会误解。^①

在详细设计层面使用构造型帮助识别构件的特性也很有价值。例如, `<<infrastructure>>` 可以用来标识基础设施构件; `<<database>>` 可以用来标识服务于一个或多个设计类或者整个系统的数据库; `<<table>>` 可以用来标识数据库中的表。

接口。接口提供关于通信和协作的重要信息(也可以帮助我们实现 OCP 原则)。然而, 接口表示的随意性会使构件图趋于复杂化。Ambler[Amb02c] 建议: (1) 当构件图变得复杂时, 在较正式的 UML 框和虚箭头记号方法中使用接口的棒棒糖式记号^②; (2) 为了保持一致, 接口都放在构件框的左边; (3) 即使其他的接口也适用, 也只表示出那些与构件相关的接口。这些建议意在简化 UML 构件图, 使其易于查看。

依赖与继承。为了提高可读性, 依赖关系自左向右, 继承关系自底(导出类)向上(基类)。另外, 构件之间的依赖关系通过接口来表示, 而不是采用“构件到构件”的方法来表示。遵照 OCP 的思想, 这种方法使得系统更易于维护。

14.2.3 内聚性

在第 12 章中, 我们将内聚性描述为构件的“专一性”。在面向对象系统进行构件级设计时, 内聚性意味着构件或者类只封装那些相互关联密切, 以及与构件或类自身有密切关系的属性和操作。Lethbridge 和 Laganière[Let01] 定义了许多不同类型的内聚性(按照内聚性的级别排序^③)。

功能内聚。主要通过操作来体现, 当一个模块完成一组且只有一组操作并返回结果时, 就称此模块是功能内聚的。

分层内聚。由包、构件和类来体现。高层能够访问低层的服务, 但低层不能访问高层的服务。例如, 如果警报响起, SafeHome 的安全功能需要打出一个电话。可以定义如图 14-5 所示的一组分层包, 带阴影的包中包含基础设施构件。访问都是从 Control panel 包向下进行的。

通信内聚。访问相同数据的所有操作被定义在一个类中。一般说来, 这些类只着眼于数据的查询、访问和存储。

那些体现出功能、层和通信等内聚性的类和构件, 相对来说易于实现、测试和维护。设计者应该尽可能获得这些级别的内聚性。然而, 需要强调的是, 实际的设计和实现问题有时会迫使设计者选择低级别的内聚性。

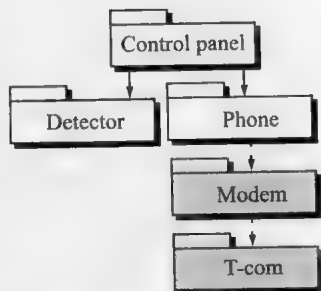


图 14-5 分层内聚

建议 尽管理解各种级别的内聚性很有益, 但是更为重要的是在设计构件时对内聚性有全面的理解。尽可能保持高内聚性。

SafeHome 内聚性的应用

[场景] Jamie 的工作间。

[人物] Jamie 和 Ed, SafeHome 软件工程团队成员, 正在实现监视功能。

[对话]

Ed: 我已经完成了 camera (摄像机) 构件的初步设计。

① 由销售部和顾客部(非技术类型)人员检查详细的设计信息是靠不住的。

② 这里指的是类似图 14-1 中的接口表示。——译者注

③ 一般来说, 内聚性级别越高, 构件实现、测试和维护起来就越容易。

Jamie: 希望快速评审一下吗?

Ed: 我想……但实际上,你最好输入一些信息。(Jamie 示意他继续。)

Ed: 我们起初为 camera 构件定义了 5 种操作。看……

determineType() 给出用户摄像机的类型。

translateLocation() 允许用户删除设计图上的摄像机。

displayID() 得到摄像机 ID,并将其显示在摄像机图标附近。

displayView() 以图形化方式给出用户摄像机的视角范围。

displayZoom() 以图形化方式给出用户摄像机的放大率。

Ed: 我分别进行了设计,并且它们非常易于操作。所以我认为,将所有的显示操作集成到一个称为 displayCamera() 的操作中——它可以显示 ID、视角和放大率等,是一个不错的主意。你不这样认为吗?

Jamie (扮了个鬼脸): 我不敢肯定。

Ed (皱眉): 为什么?所有这些小操作是很令人头疼的!

Jamie: 问题是当将它们集成到一起

后,我们将失去内聚性。你知道的,displayCamera() 操作不是专一的。

Ed (略有不快): 那又怎样?这样做使得我们最多只需不到 100 行的源代码。我想实现起来也比较简单。

Jamie: 如果销售人员决定更改我们显示视角域的方式怎么办?

Ed: 我马上跳到 displayCamera() 操作,并进行这种改变。

Jamie: 那么引起的副作用怎么办?

Ed: 什么意思?

Jamie: 也就是说你做了修改,但是不经意之间产生了 ID 的显示问题。

Ed: 我没有那么笨。

Jamie: 可能没有,但是如果两年以后某些支持人员必须做这种改变怎么办。他可能并不像你一样理解这个操作,谁知道呢,也许他很粗心。

Ed: 所以你反对?

Jamie: 你是设计师,这是你的决定,只要确信你理解了低内聚性的后果。

Ed (想了想): 可能我们要设计一个单独的显示操作。

Jamie: 好主意。

[297]

14.2.4 耦合性

在前面关于分析和设计的讨论中,我们知道通信和协作是面向对象系统中的基本要素。然而,这个重要(必要)特征存在一个黑暗面。随着通信和协作数量的增长(也就是说,随着类之间的联系程度越来越强),系统的复杂性也随之增长了。同时,随着系统复杂度的增长,软件实现、测试和维护的困难也随之增大。

耦合是类之间彼此联系程度的一种定性度量。随着类(构件)之间的相互依赖越来越多,类之间的耦合程度亦会增加。在构件级设计中,一个重要的目标就是尽可能保持低耦合。

有多种方法来表示类之间的耦合。Lethbridge 和 Laganière[Let01] 定义了一组耦合分类。例如,内容耦合发生在当一个构件“暗中修改其他构件的内部数据”[Let01]时。这违反了基本设计概念当中的信息隐蔽原则。控制耦合发生在当操作 A 调用操作 B,并且向 B 传递了一个控制标记时。接着,控制标记将会指引 B 中的逻辑流程。这种耦合形式的主要问题在于,B 中的一个不相关变更往往能够导

建议 随着每个软件构件的细化,我们的关注点将转移到数据结构设计上,包括其相关的过程设计。但是,不要忘记体系结构,因为构件和服务于构件的全局数据结构都住在体系结构这所大房子里。

致 A 所传递控制标记的意义也必须发生变更。如果忽略这个问题，就会引起错误。外部耦合发生在当一个构件和基础设施构件（例如，操作系统功能、数据库容量、无线通信功能等）进行通信和协作时。尽管这种类型的耦合是必要的，但是在一个系统中应该尽量将这种耦合限制在少量的构件或者类范围内。

软件必须进行内部和外部的通信，因此，耦合是必然存在的。然而，在不可避免出现耦合的情况下，设计者应该尽力降低耦合性，并且要充分理解高耦合的后果。

SafeHome 耦合的应用

[场景] Shakira 的工作间。

[人物] Vinod 和 Shakira, SafeHome 软件工程团队成员，正在实现安全功能。

[对话]

Shakira：我曾经有一个非常好的想法，但之后我又考虑了一下，觉得好像并没有那么好。最后我还是放弃了，不过我想最好和你讨论一下。

Vinod：当然可以，是什么想法？

Shakira：好的，每个传感器能够识别一种警报条件，对吗？

Vinod（微笑）：这就是我们称它为传感器的一个原因啊，Shakira。

Shakira（恼怒）：你讽刺我！你应该好好学习一下处理人际关系的技巧。

Vinod：你刚才说什么？

Shakira：我指的是……为什么不为每个传感器都创建一个名为 `makeCall()` 的操作，该操作能够直接和 `OutgoingCall`（外呼）

构件协作，也就是通过 `OutgoingCall` 构件的接口实现协作？

Vinod（沉思着）：你的意思是让协作发生在 `ControlPanel` 之类的构件之外？

Shakira：是的，但接着我又对自己说，这将意味着每个传感器对象都会与 `OutgoingCall` 构件相关联，而这意味着与外部世界的间接耦合……我想这样会使事情变得复杂。

Vinod：我同意，在这种情况下，让传感器接口将信息传递给 `ControlPanel`，并且让其启动外呼，这是一个比较好的主意。此外，不同的传感器将导致不同的电话号码。在信息改变时，你并不希望传感器存储这些信息，因为如果发生变化……

Shakira：感觉不太对。

Vinod：耦合设计方法告诉我们是有点不对。

Shakira：无论如何……

298

14.3 实施构件级设计

在本章的前半部分，我们已经知道构件级设计本质上是精细化的。设计者必须将需求模型和架构模型中的信息转化为一种设计表示，这种表示提供了用来指导构件（编码和测试）活动的充分信息。应用于面向对象系统时，下面的步骤表示出构件级设计典型的任务集。

步骤 1：标识出所有与问题域相对应的设计类。使用需求模型和架构模型，正如 14.1.1 节中所描述的那样，每个分析类和体系结构构件都要细化。

步骤 2：确定所有与基础设施域相对应的设计类。在需求模型中并没有描述这些类，并且在体系结构设计中也经常忽略这些类，但是此时必须对它们进行描述。如前所述，这种类型的类和构件包括 GUI（图形用户界面）构件（通常为可复用构件）、操作系统构件以及对象

引述 如果我有更多的时间，我将写一封更短的信。

Blaise Pascal

和数据管理构件等。

步骤 3：细化所有不需要作为复用构件的设计类。详细描述实现类细化所需要的所有接口、属性和操作。在实现这个任务时，必须考虑采用设计的启发式规则（如构件的内聚和耦合）。

步骤 3a：在类或构件协作时说明消息的细节。需求模型中用协作图来显示分析类之间的相互协作。在构件级设计过程中，某些情况下有必要通过对系统中对象间传递消息的结构进行说明来表现协作细节。尽管这是一个可选的设计活动，但是它可以作为接口规格说明的前提，这些接口显示了系统构件之间通信和协作的方式。

建议 如果在非面向对象的环境下工作，那么前 3 步的重点在于提炼数据对象和处理功能（转换），这些功能被视为需求模型的一部分。

图 14-6 给出了前面提到的影印系统的一个简单协作图。ProductionJob、WorkOrder 和 JobQueue 这三个对象相互协作，为生产线准备印刷作业。图中的箭头表示对象间传递的消息。在需求建模时，消息说明如图 14-6 所示。然而，随着设计的进行，消息通过下列方式的扩展语法来细化 [Ben02]：

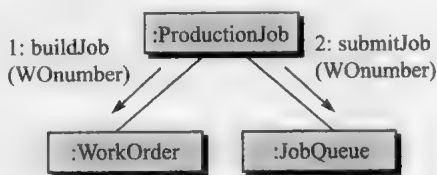


图 14-6 带消息的协作图

```
[guard condition] sequence expression (return value) :=
message name (argument list)
```

其中，[guard condition] 采用对象约束语言（Object Constraint Language, OCL）^①来书写，并且说明了在消息发出之前应该满足什么样的条件集合；sequence expression 是一个表明消息发送序号的整数（或其他样式的表明发送顺序的指示符，如 3.1.2）；(return value) 是由消息唤醒操作返回的信息名；message name 表示唤醒的操作，(argument list) 是传递给操作的属性列表。

步骤 3b：为每个构件确定适当的接口。在构件级设计中，一个 UML 接口是“一组外部可见的（即公共的）操作。接口不包括内部结构，没有属性，没有关联……”[Ben02]。更正式地讲，接口就是某个抽象类的等价物，该抽象类提供了设计类之间的可控连接。图 14-1 给出了接口细化的实例。实际上，为设计类定义的操作可以归结为一个或者多个抽象类。抽象类内的每个操作（接口）应该是内聚的，也就是说，它应该展示那些关注于一个有限功能或者子功能的处理。

参照图 14-1，initiateJob 接口由于没有展现出足够的内聚性而受到争议。实际上，它完成了三个不同的子功能：建立工作单，检查任务的优先级，并将任务传递给生产线。接口设计应该重构。一种方法就是重新检查设计类，并定义一个新类 WorkOrder，该类的作用就是处理与装配工作单相关的所有活动。操作 buildWorkOrder() 成为该类的一部分。类似地，我们可能需要定义包括操作 checkPriority() 在内的 JobQueue 类。ProductionJob 类包括给生产线传递生产任务的所有相关信息。initiateJob 接口将采用图 14-7 所示的形式。initiateJob 现在是内聚的，集中在一个功能上。与 ProductionJob、WorkOrder 和 JobQueue 相关的接口几乎都是专一的。

步骤 3c：细化属性，并且定义实现属性所需要的数据类型和数据结构。描述属性的数据类型和数据结构一般都需要在实现时所采用的程序设计语言中进行定义。UML 采用下面

① OCL 在附录 1 中有简明的介绍。

的语法来定义属性的数据类型：

```
name:type-expression = initial-value{property string}
```

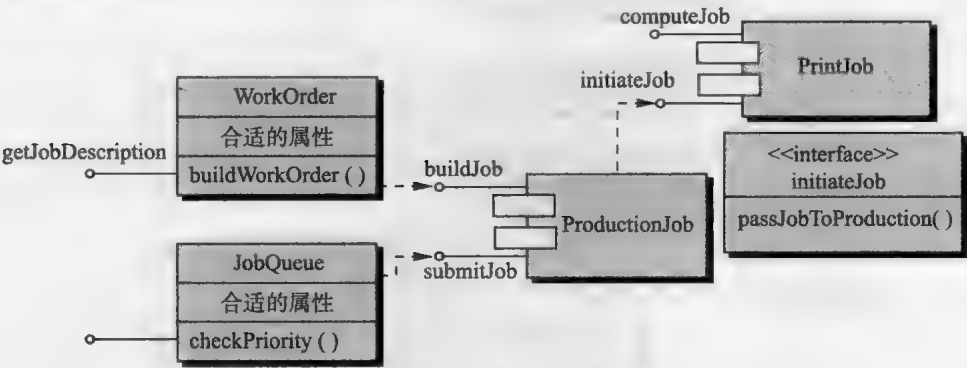


图 14-7 为 PrintJob 重构接口和类定义

其中，name 是属性名，type-expression 是数据类型，initial-value 是创建对象时属性的取值，property string 用于定义属性的特征或特性。

301

在构件级设计的第一轮迭代中，属性通常用名字来描述。再次参考图 14-1，PrintJob 的属性列表只列出了属性名。然而，随着设计的进一步细化，我们将使用 UML 的属性格式注释来定义每个属性。例如，以下列方式来定义 paperType-weight：

```
paperType-weight: string = "A" {contains 1 of 4 values-A, B, C, or D}
```

这里将 paperType-weight 定义为一个字符串变量，初始值为 A，可以取值为集合 {A, B, C, D} 中的一个。

如果某一属性在多个设计类中重复出现，并且其自身具有比较复杂的结构，那么最好是为这个属性创建一个单独的类。

步骤 3d：详细描述每个操作中的处理流。这可能需要由基于程序设计语言的伪代码或者由 UML 活动图来完成。每个软件构件都需要应用逐步求精概念（第 12 章）通过很多次迭代进行细化。

第一轮迭代中，将每个操作都定义为设计类的一部分。在任何情况下，操作应该确保具有高内聚性的特征；也就是说，一个操作应该完成单一的目标功能或者子功能。接下去的一轮迭代，只是完成对操作名的详细扩展。例如，图 14-1 中的操作 computePaperCost() 可以采用如下方式进行扩展：

```
computePaperCost(weight, size, color): numeric
```

这种方式说明 computPageCost() 要求属性 weight、size 和 color 作为输入，并返回一个数值（实际上为金额）作为输出。

如果实现 computePaperCost() 的算法简单而且易于理解，则没有必要开展进一步的设计细化。软件编码人员将会提供实现这些操作的必要细节。但是，如果算法比较复杂或者难于理解，此时则需要进一步的设计细化。图 14-8 给出了操作 computePaperCost() 的 UML 活动图。当活动图用于构件级设计的规格说明时，通常都在比源码更高的抽象级上表示。还有一种方法是，在设计规格说明中使用伪代码，这部分内容将在 14.5.3 节进行讨论。

建议 在细化构件设计时使用逐步求精的方法。经常提出这样的问题：“是否存在一种方法可以简化问题并仍能达到相同的结果？”

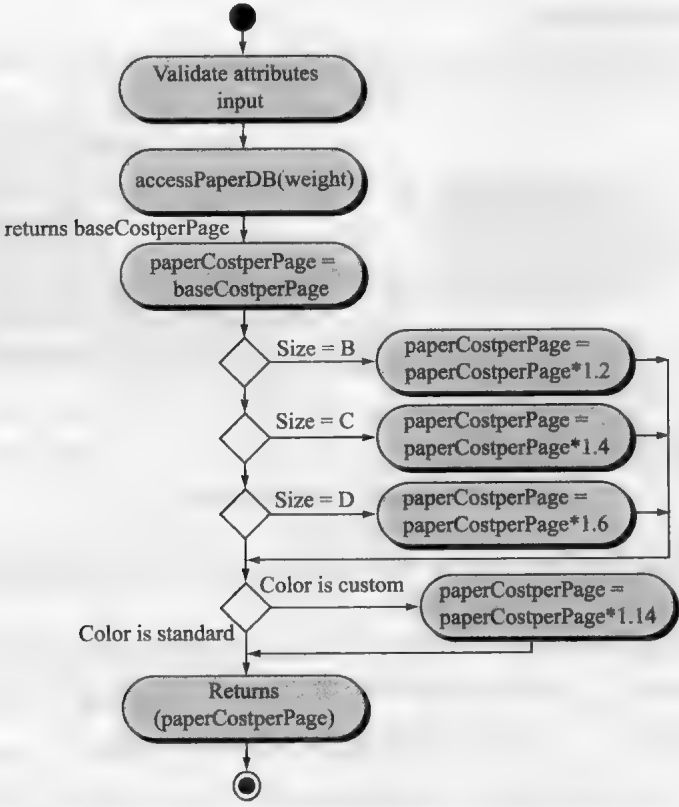


图 14-8 computePaperCost() 操作的 UML 活动图

步骤 4：说明持久数据源（数据库和文件）并确定管理数据源所需要的类。数据库和文件通常都凌驾于单独的构件设计描述之上。在多数情况下，这些持久数据存储最初都作为体系结构设计的一部分进行说明，然而，随着设计细化过程的不断深入，提供关于这些持久数据源的结构和组织等额外细节常常是有用的。

步骤 5：开发并且细化类或构件的行为表示。UML 状态图被用作需求模型的一部分，表示系统的外部可观察的行为和更多的分析类个体的局部行为。在构件级设计过程中，有些时候对设计类的行为进行建模是必要的。

对象（程序执行时的设计类实例）的动态行为受到外部事件和对象当前状态（行为方式）的影响。为了理解对象的动态行为，设计者必须检查设计类生命周期中所有相关的用例，这些用例提供的信息可以帮助设计者描述影响对象的事件，以及随着时间流逝和事件的发生对象所处的状态。图 14-9 描述了使用 UML 状态图 [Ben02] 表示的状态之间的转换（由事件驱动）。

从一种状态到另一种状态的转换（用圆角矩形来表示），都表示为如下形式的事件序列：

```
event-name (parameter-list) [guard-condition]/action expression
```

其中，event-name 表示事件；parameter-list 包含了与事件相关的数据；guard-condition 采用对象约束语言 OCL 书写，并描述了事件发生前必须满足的条件；action expression 定义了状态转换时发生的动作。

参照图 14-9，针对状态的进入和离开两种情形，每个状态都可以定义 entry/ 和 exit/ 两个动作。在大多数情况下，这些动作与正在建模的类的相关操作相对应。do/ 指示符提供了

302
303

一种机制，用来显示伴随此种状态的相关活动；而 include/ 指示符则提供了通过在状态定义中嵌入更多状态图细节的方式进行细化的手段。

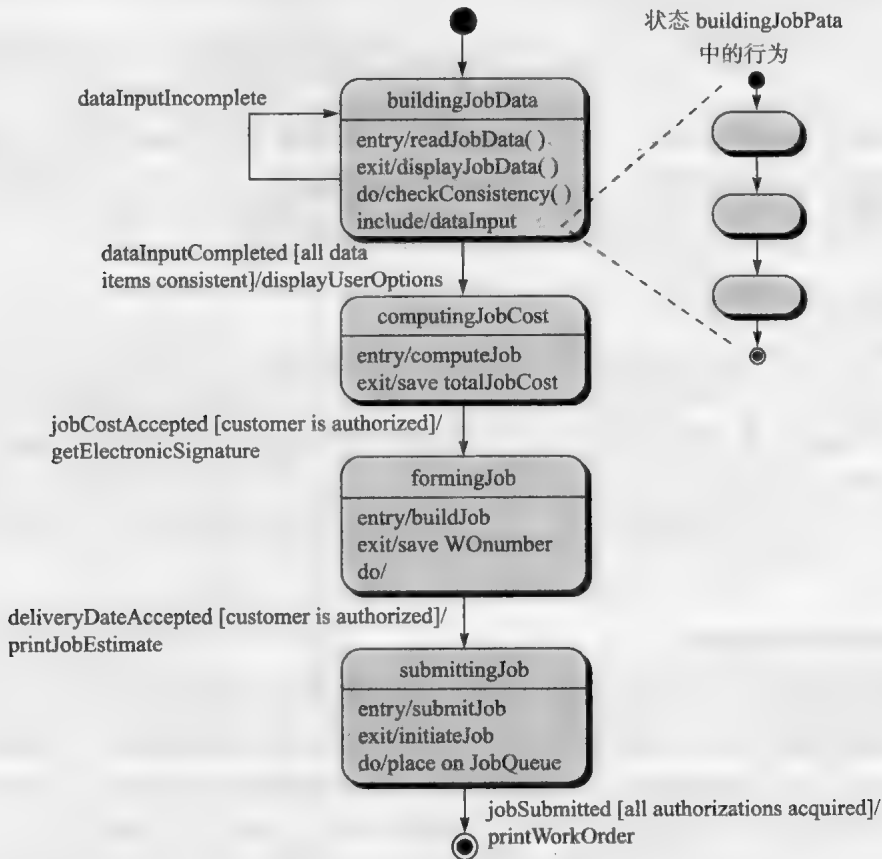


图 14-9 PrintJob 类的状态图

需要注意的重要一点是，行为模型经常包含一些在其他设计模型中不明显的信息。例如，通过仔细查看图 14-9 中的状态图可以知道，当得出印刷任务的成本和进度数据时，PrintJob 类的动态行为取决于用户对此是否认可。如果没有得到允许（警戒条件确保用户有权审核），印刷工作就不能提交，因为不可能到达 submittingJob 状态。 [304]

步骤 6：细化部署图以提供额外的实现细节。部署图（第 12 章）作为体系结构设计的一部分，采用描述符形式来表示。在这种表示形式中，主要的系统功能（经常表现为子系统）都表示在容纳这些功能的计算环境中。

在构件级设计过程中，应该对部署图进行细化，以表示主要构件包的位置。然而，一般在构件图中不单独表示构件，目的在于避免图的复杂性。某些情况下，部署图在这个时候被细化成实例形式。这意味着要对指定的硬件和使用的操作系统环境加以说明，而构件包在这个环境中的位置也需要确定。

步骤 7：考虑每个构件级设计表示，并且时刻考虑其他可选方案。纵观全书，我们始终强调设计是一个迭代过程。创建的第一个构件级模型总没有迭代多次之后得到的模型那么全面、一致或精确。在进行设计工作时，重构是十分必要的。

另外，设计者不能眼光狭隘。设计中经常存在其他的设计方案，在没有决定最终设计模型之前，最好的设计师会考虑所有（或大部分）的方案，运用第 12 章和本章介绍的设计原

则和概念开发其他的可选方案，并且仔细考虑和分析这些方案。

14.4 WebApp 的构件级设计

在基于 Web 的系统和应用中，内容和功能的界限常常是模糊的。因此有必要考虑什么是 WebApp 构件。

根据本章的内容可以知道，WebApp 构件是：（1）定义良好的聚合功能，为最终用户处理内容，或提供计算或数据处理；（2）内容和功能的聚合包，提供最终用户所需的功能。因此 WebApp 构件级设计通常包括内容设计元素和功能设计元素。

[305]

14.4.1 构件级内容设计

构件级内容设计关注内容对象，以及包装后展示给 WebApp 最终用户的方式。构件级内容设计应该适合创建的 WebApp 的特性。在很多情况下，内容对象不需要被组织成构件，它们可以分别实现。但是，随着 WebApp、内容对象以及它们之间相互关系的规模和复杂度的增长，在更好的参考和设计方法下组织内容是十分必要的^①。此外，如果内容显示出高度的动态性（如一个在线拍卖网站的内容），那么建立一个包含内容构件的、清晰的结构模型是非常重要的。

14.4.2 构件级功能设计

WebApp 是作为一系列构件交付的，这些构件与信息体系结构并行开发，以确保它们的一致性。最重要的是，在一开始就要考虑需求模型和初始信息体系结构，然后再进一步考查功能如何影响用户与系统的交互、要展示的信息以及要管理的用户任务。

在体系结构设计中，往往将 WebApp 的内容和功能结合在一起来设计应用系统的功能体系结构。在这里，功能体系结构代表的是 WebApp 的功能域，并且描述了关键的功能构件和这些构件是如何进行交互的。

14.5 移动 App 的构件级设计

在第 13 章中我们指出，移动 App 通常使用多层结构，包括用户界面层、业务层和数据层。如果你正在为一个基于 Web 的瘦客户端构建移动 App，那么只有那些实现用户界面所需的构件才会驻留在移动设备上。在移动设备上，一些应用可能包含用于实现业务层和数据层需求的构件，这些层会受到设备物理特性的限制。

首先考虑用户界面层，重要的一点是，较小的显示区域要求设计者必须选择要显示的内容（文本和图形）。这样可能有助于为特定的用户组定制不同内容，且仅显示每组所需要的内容。业务层和数据层通常是由 Web 或云服务构件来实现的。如果提供业务和数据服务的构件完全驻留在移动设备上，那么连接问题并不是需要重点关注的问题。设计时必须考虑在网络连接中断（或丢失）时，构件如何访问当前应用驻留在网络服务器上的数据。

[306]

将桌面应用移植到移动设备上时，需要对业务层构件进行检查，看它们是否满足新平台所需的非功能性需求（例如，安全性、性能、可访问性）。目标移动设备可能缺乏必要的处理器速度、内存或显示性能。在第 18 章将更详细地介绍移动 App 的设计。

① 内容构件可以在其他 WebApp 中复用。

14.6 设计传统构件

传统软件构件^①的构件级设计基础在20世纪60年代早期已经形成，在Edsger Dijkstra ([Dij65], [Dij76b]) 及他人的著作（如 [Boh66]）中又得到了进一步的完善。20世纪60年代末，Dijkstra等人提出，所有的程序都可以建立在一组限定好的逻辑构造上。这组逻辑构造强调“对功能域的支持”，其中每个逻辑结构都是可预测的，过程流从顶端进入，从底端退出，读者很容易理解。

这些结构包括顺序型、条件型和重复型。顺序型实现了任何算法规格说明中的核心处理步骤；条件型允许根据逻辑情况选择处理的方式；重复型提供了循环。这三种结构是结构化程序设计的基础，而结构化程序设计是一种重要的构件级设计技术。

结构化的构建使得软件的过程设计只采用少数可预知的逻辑结构。复杂性度量（第30章）表明，使用结构化的构造降低了程序复杂性，从而增加了可读性、可测试性和可维护性。使用有限数量的逻辑结构也符合心理学家所谓的人类成块的理解过程。要理解这一过程，可以考虑阅读英文的方式。读者不是阅读单个字母，而是辨认由单词或短语构成的模式或是字母块。结构化的构造就是一些逻辑块，读者可以用它来辨认模块的过程元素，而不必逐行阅读设计或是代码，当遇到了容易辨认的逻辑模式时，理解力就得到了提高。

无论是对于应用领域还是对于技术复杂度，任何程序都可以只用这三种结构化的构造来设计和实现。然而，需要注意的是，教条地使用这些结构在实践中会遇到困难。

关键点 结构化程序设计是一种设计技术，该技术将程序逻辑流程限制为以下三种结构：顺序型，条件型和重复型。

307

14.7 基于构件的开发

在软件工程领域，复用既是老概念，也是新概念。在计算机发展的早期，程序员就已经复用概念、抽象和过程，但是早期的复用更像是一种临时的方法。今天，复杂的、高质量的计算机系统往往必须在短时间内开发完成，所以需要一种更系统的、更有组织性的复用方法来协助这样的快速开发。

基于构件的软件工程 (Component-Based Software Engineering, CBSE) 是一种强调使用可复用的软件构件来设计与构造计算机系统的过程。考虑到这种解释，很多问题出现了，仅仅将多组可复用的软件构件组合起来，就能够构造出一个复杂的系统吗？这种工作能够以一种高效和节省成本的方式完成吗？能否建立恰当的激励机制来鼓励软件工程师复用而不是重复开发？管理团队是否也愿意为构造可复用软件构件过程中的额外开销买单？能否以使用者易于访问的方式构造复用所必需的构件库？已有的构件可以被需要的人找到并使用吗？大家渐渐地发现了这些问题的答案，而且这些问题的答案都是“可以”。

引述 领域工程是发现系统间的共性，以识别可以应用于很多系统的构件，并识别最能充分利用那些构件的程序族。

Paul Clements

14.7.1 领域工程

领域工程的目的是识别、构造、分类和传播一组软件构件，这些构件

① 传统的软件构件实现处理元素，这些处理元素涉及问题域中的功能或子功能，或者涉及基础设施域中的某种性能。通常将传统构件称为模块、程序或子程序，传统构件不像面向对象构件那样封装数据。

在某一特定的应用领域中可以适用于现有和未来的软件^①。总体目标是为软件工程师建立一种机制来分享这些构件，从而在开发新系统或改造现有系统时可以共享这些构件——复用它们。领域工程包括三种主要活动：分析、构建和传播。

领域分析的总体方法通常在面向对象软件工程的环境中赋予特色。领域分析过程中的步骤为：（1）定义待研究的领域；（2）对从领域中提取的项进行分类；（3）收集领域中有代表性的应用系统样本；（4）分析样本中的每个应用，并且定义分析类；（5）为这些类开发需求模型。值得注意的是，领域分析适用于任何软件工程范型，因此，领域分析可以应用到传统的软件开发和面向对象的软件开发中。

建议 这一节所讨论的分析过程主要集中于可复用构件。但是，完整的COTS系统（例如，电子商务应用、自动销售应用）分析也可以是领域分析的一部分。

14.7.2 构件的合格性检验、适应性修改与组合

领域工程提供了基于构件的软件工程（CBSE）所需的可复用构件库。某些可复用构件是自行开发的，有些可以从现有的系统中抽取得到，还可以从第三方获得。

遗憾的是，可复用构件的存在并不能保证这些构件可以很容易或很有效地被集成到为新应用所选择的体系结构中。正因为如此，当计划使用某一构件时，要进行一系列的基于构件的开发活动。

构件合格性检验。构件合格性检验将保证某候选构件能够执行需要的功能，完全适合系统的体系结构（第13章），并具有该应用所需的质量特性（例如，性能、可靠性、可用性）。

契约式设计这种技术着重于定义明确的和可核查的构件接口规格说明，从而使构件的潜在用户快速了解其意图。我们将称为前置条件、后置条件和不变式的表述加入到构件规格说明中^②。表述使得开发人员知道构件提供什么功能，以及它在一定条件下的行为方式。这种表述使开发人员更容易识别合格的构件，因而更愿意在他们的设计中信任和使用这些构件。当构件有一个“经济型接口”时，契约式设计就得到了增强。构件接口具有一组最小的必要操作，使其能够完成职责（契约）。

接口规格说明提供了有关软件构件的操作和使用的有用信息，但是，对于确定该构件是否能在新的应用系统中高效复用，它并未提供需要的所有信息。这里列出了构件合格性检验的一些重要因素 [Bro96]：

- 应用编程接口（Application Programming Interface，API）。
- 构件所需的开发工具与集成工具。
- 运行时需求，包括资源使用（如内存或外存储器）、时间或速度以及网络协议。
- 服务需求，包括操作系统接口和来自其他构件的支持。
- 安全特征，包括访问控制和身份验证协议。
- 嵌入式设计假设，包括特定的数值或非数值算法的使用。
- 异常处理。

计划使用自行开发的可复用构件时，这些因素都是比较容易评估的。如果构件开发过程

提问 构件合格性的重要因素有哪些？

① 第13章中曾经提到识别特定应用领域的体系结构类型。

② 前置条件是对构件使用之前必须做验证的假设所作的叙述，后置条件是对将要交付的构件可提供的服务保证的叙述，不变式是对不会被构件变更的系统属性的叙述。这些概念将在第28章介绍。

应用了良好的软件工程实践,那么之前列出的问题就都容易回答。但是,要确定商业成品构件(Commercial Off-The-Shelf, COTS)或第三方构件的内部工作细节就比较困难了,因为能够得到的信息可能只有接口规格说明。

构件适应性修改。在理想情况下,领域工程建立构件库,构件可以很容易地被集成到应用体系结构中。“容易集成”的含义是:(1)对于库中的所有构件,都已经实现了一致的资源管理方法;(2)所有构件都存在诸如数据管理等公共活动;(3)已经以一致的方式实现了体系结构的内部接口及与外部环境的接口。

实际上,即使已经对某个构件在应用体系结构内部的使用进行了合格性检验,也可能在刚才提到的一个或多个地方发生冲突。为了避免这些冲突,经常使用一种称为构件包装[Bro96]的适应性修改技术。当软件团队对某一构件的内部设计和代码具有完全的访问权时(通常不是这样,除非使用开源的COTS构件),则可应用白盒包装技术。与软件测试中白盒测试(第23章)相对应,白盒包装检查构件的内部处理细节,并进行代码级的修改来消除所有冲突。当构件库提供了能够消除或掩盖冲突的构件扩展语言或API时,可以应用灰盒包装技术。黑盒包装技术需要在构件接口中引入预处理和后处理来消除或掩盖冲突。软件团队需要决定是否值得充分包装构件,或者考虑是否开发定制构件(对构件进行设计以消除遇到的冲突)。

建议 软件团队

除了需要评估为复用所做的适应性修改是否是成本合算的,还需要评估取得所需要的功能和性能是否也是成本合算的。

构件组合。构件组合任务将经过合格性检验、适应性修改以及工程开发的构件组合到为应用建立的体系结构中。为完成这项任务,必须建立一个基础设施以将构件绑定到一个运行系统中。该基础设施(通常是专门的构件库)提供了构件协作的模型和使构件能够相互协作并完成共同任务的特定服务。

310

由于复用和CBSE对软件业影响巨大,因此大量的主流公司和产业协会已经提出了软件构件标准^①。这些标准包括:CCM(Corba Component Model, Corba 构件模型)^②, Microsoft COM 和 .NET^③, JavaBeans^④, 以及 OSGI (Open Services Gateway Initiative 开放服务网关协议[OSGI3])^⑤。这些标准中没有哪一种在产业界独占优势。虽然很多开发者已经采用了其中的某个标准,但大型软件组织仍可能根据应用的类型和采用的平台来选择使用某种标准。

14.7.3 体系结构不匹配

广泛复用所面临的挑战之一就是体系结构不匹配[Gar09a]。可复用的构件设计者常常对耦合构件的有关环境进行隐式假设。这些假设往往侧重于构件控制模型、构件的连接属性(接口)、体系结构基础设施以及体系结构构建过程中的性质。如果这些假设是不正确的,就产生了体系结构不匹配的情况。

设计概念,如抽象、隐蔽、功能独立、细化和结构化程序设计、面向对象的方法、测

① 为了实现CBSE,过去和现在工业界都做了很多工作,Greg Olesn[Ols06]对此给出了精彩的讨论。Ivica Crnkovic [Crk11]给出了关于更多近期的工业构件模型的探讨。

② 关于CCM的进一步资料可在www.omg.org找到。

③ 关于COM和.NET的资料,可在www.microsoft.com/COM及msdn2.microsoft.com/en-us/netframework/default.aspx找到。

④ 关于javabean的最新资料可在java.sun.com/product/javabeans/docs/找到。

⑤ 关于OSGI的资料可在<http://www.osgi.org/Main/Homepage>找到。

试、软件质量保证 (SQA) 和正确性验证方法 (第 28 章), 所有这些都有助于创建可复用的软件构件和防止体系结构不匹配。

如果利益相关者的设想得到了明确的记载, 那么在早期就能发现体系结构不匹配。此外, 风险驱动过程模型的使用强调了早期体系结构原型的定义, 并且指出了不匹配的区域。如果不采取一些诸如封装或适配器[⊖]的机制, 往往很难修复体系结构不匹配。有时甚至需要通过完全重新设计构件接口或者通过构件本身来消除耦合问题。

[311]

14.7.4 复用的分析与设计

将需求模型 (第 9 ~ 11 章) 中的元素与可复用构件描述进行比较的过程有时称为“规格说明匹配” [Bel95]。如果规格说明匹配指出一个现有的构件和现有应用需求相符合, 那么就可以从可复用构件库中提取这些构件, 并将它们应用于新系统的设计中。如果没有发现这样的构件 (即没有匹配), 就需要创建新构件。在这一点上, 当需要建立新构件时, 应该考虑可复用性设计 (Design for reuse, DFR)。

正如我们已经谈到的, DFR 需要软件工程师采用良好的软件设计概念和规则 (第 12 章)。但是, 也必须考虑应用领域的特点。Binder [Bin93] 提出了构成可复用设计基础的一系列关键问题[⊖]。如果应用领域定义了标准的全局数据结构, 则应使用这些标准的数据结构来设计构件。在应用领域内应采用标准接口协议, 并且可选取一种适合应用领域的体系结构风格 (第 13 章) 作为新软件体系结构设计的模板。一旦建立了标准数据、接口和程序模板, 就有了进行设计所依托的框架。符合此框架的新构件将来复用的可能性就比较高。

建议 如果构件必须与遗留系统及多个系统 (它们的体系结构和接口协议不一致) 进行接口或者集成, DFR 可能会非常困难。

14.7.5 构件的分类与检索

考虑一个大型构件库, 其中存放了成千上万的可复用构件。但是, 软件工程师如何才能找到他所需要的构件呢? 为了回答这个问题, 又出现了另一个问题: 我们如何以无歧义的、可分类的术语来描述软件构件? 这些问题太难, 至今还没有明确答案。

可以用很多种方式来描述可复用软件构件, 但是理想的描述应包括 Tracz [Tra95] 提出的 3C 模型——概念 (concept)、内容 (content) 和环境 (context), 即描述构件能够实现什么功能, 如何实现那些对一般用户来讲是隐蔽的内容 (只有那些想要修改或测试该构件的人才需要了解), 以及将可复用软件构件放到什么样的应用领域中。

为了在实际环境中使用, 概念、内容和环境必须被转换为具体的规格说明模式。关于可复用软件构件分类模式的文章很多 (例如, [Nir10], [Cec06]), 所有这些分类模式都应该在可复用环境中实现, 该环境应具备以下几方面的特点:

- 能够存储软件构件和检索该构件所需分类信息的构件数据库。
- 提供访问数据库的管理系统。
- 包含软件构件检索系统 (例如对象请求代理), 允许客户应用从构件库服务器中检索构件和服务。

提问 构件复用环境的关键特性是什么?

[312]

⊖ 适配器是一种软件设备, 它允许具有不匹配接口的客户端访问构件, 方法是 将服务请求翻译成可以访问原始接口的形式。

⊖ 一般来说, DFR 准备工作应当是领域工程的一部分。

- 提供 CBSE 工具，支持将复用的构件集成到新的设计或实现中。

每种功能都与复用库交互，或是嵌入在复用库中。复用库是更大型软件库（第 29 章）的一个元素，并且为软件构件及各种可复用的工作产品（例如，规格说明、设计、模式、框架、代码段、测试用例、用户指南）提供存储设施。

软件工具 CBSE

[目标] 在软件构件的建模、设计、评审以及集成成为更大系统的一部分时起辅助作用。

[机制] 工具的机制各异。一般情况下，CBSE 工具对以下一项或多项工作起辅助作用：软件体系结构的规格说明和建模，可利用的软件构件的浏览及选择，构件集成。

[代表性工具]^①

- ComponentSource (www.componentsource.com)。提供了大量被许多不同构件标准支持的 COTS 软件构件（及工具）。
- Component Manager。由 Flashline ([http://www.softlookup.com/download.asp?](http://www.softlookup.com/download.asp?id=8204)

[id=8204](http://www.softlookup.com/download.asp?id=8204)) 开发，“它是一个应用程序，能支持、促进及测量软件构件复用”。

- Select Component Factory。由 Select Business Solution (www.selectbs.com) 开发，“它是一个集成的成套产品，用于软件设计、设计审查、服务/构件管理、需求管理及代码生成”。
- Software Through Pictures-ACD。由 Aonix (www.aonix.com) 发布，对于由 OMG 模型驱动的体系结构（一个开放的、供供应商中立的 CBSE 方法），它能够运用 UML 进行广泛的建模。

14.8 小结

构件级的设计过程包含一系列活动，这些活动逐渐降低了软件表示的抽象层次。构件级设计最终在接近于代码的抽象层次上描述软件。

根据所开发软件的特点，可以从三个不同的角度来进行构件设计。面向对象的视角注重细化来自于问题域和基础设施域的设计类。传统的视角细化三种不同的构件或模块：控制模块、问题域模块和基础设施模块。在这两种视角中，都需要应用那些能够得到高质量软件的基本设计原则和概念。从过程视角考虑时，构件设计采用了可复用的软件构件和设计模式，这些都是基于构件级的软件工程的关键要素。

在进行类的细化时，有许多重要的原则和概念可以指导设计者，包括开闭原则、依赖倒置原则，以及耦合性和内聚性概念等，这些都可以指导工程师构造可测试、可实现和可维护的软件构件。在这种情况下，为了实施构件级设计，需要细化类，这可通过以下方式达到：详细描述消息细节，确定合适的接口，细化属性并定义实现它们的数据结构，描述每个操作的处理流程，在类或构件层次上表示行为等。在任何情况下，设计迭代（重构）都是重要活动。

传统的构件级设计需要足够详细地表示出程序模块的数据结构、接口和算法，以指导用

① 这里提到的工具只是此类工具的例子，并不代表本书支持选择采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

编程语言书写的源代码的生成。为此,设计者采用某种设计表示方法来表示构件级详细信息,可以使用图形、表格,也可以使用文本格式。

WebApp 的构件级设计既要考虑内容,也要考虑功能性,因为它是由基于 Web 系统发布的。构件级的内容设计关注展示给 WebApp 最终用户的内容对象及这些内容对象的包装方式。WebApp 的功能性设计关注处理功能,这些处理功能可以操作内容、执行计算、查询和访问数据库,并建立与其他系统的接口。所有的构件级设计原则和指导方针都适用于此。

移动 App 的构件级设计通常使用多层体系结构,包括用户界面层、业务层和数据层。如果移动 App 通常需要执行移动设备上的业务层和数据层的构件设计,那么该设备的物理硬件特性限制将成为设计上的重要约束。

结构化程序设计是一种过程设计思想,它限制描述算法细节时使用的逻辑构造的数量和类型。结构化程序设计的目的是帮助设计师定义简单的算法,使算法易于阅读、测试和维护。

基于构件的软件工程在特定的应用领域内标识、构建、分类和传播一系列软件构件。这些构件经过合格性检验和适应性修改,并集成到新系统中。对于每个应用领域,应该在建立了标准数据结构、接口协议和程序体系结构的环境中设计可复用构件。

314

习题与思考题

- 14.1 术语“构件”有时很难定义。请首先给出一个一般的定义,然后针对面向对象软件 and 传统软件给出更明确的定义,最后选择三种你熟悉的编程语言来说明如何定义构件。
- 14.2 为什么传统软件当中必要的控制构件在面向对象的软件中一般是不需要的?
- 14.3 用自己的话描述 OCP。为什么创建构件之间的抽象接口很重要?
- 14.4 用自己的话描述 DIP。如果设计人员过于依赖具体构件,会出现什么情况?
- 14.5 选择三个你最近开发的构件,并评估每个构件的内聚类型。如果要求定义高内聚的主要优点,那么主要优点会是什么?
- 14.6 选择三个你最近开发的构件,并评估每个构件的耦合类型。如果要求定义低耦合的主要优点,那么主要优点会是什么?
- 14.7 问题领域构件不会存在外部耦合的说法有道理吗?如果你认为没有道理,那么哪种类型的构件存在外部耦合?
- 14.8 完成:(1)一个细化的设计类;(2)接口描述;(3)该类中包含的某一操作的活动图;(4)前几章讨论过的某个 SafeHome 类的详细状态图。
- 14.9 逐步求精和重构是一回事吗?如果不是,它们有什么区别?
- 14.10 什么是 WebApp 构件?
- 14.11 选择已有程序的某一小部分(大概 50 ~ 75 行源代码),通过在源代码周围画框隔离出结构化编程构造。程序片段是否存在违反结构化程序设计原则的构造?如果存在,重新设计代码使其遵守结构化编程的构造,如果不违反,对于画出的框你注意到了什么?
- 14.12 所有现代编程语言都实现了结构化的程序设计构造。用三种程序设计语言举例说明。
- 14.13 选择有少量代码的构件,并使用活动图来描述它。
- 14.14 在构件级设计的评审过程中,为什么“分块”很重要?

扩展阅读与信息资源

最近几年,已经出版了许多有关基于构件的开发和构件复用的书籍。Szyperski (《Component

Software》, 2nd ed., Addison-Wesley, 2011) 强调将软件构件作为有效系统构造块的重要性。Hamlet (《Composing Software Components》, Springer, 2010)、Curtis (《Modular Web Design》, New Riders, 2009)、Apperly 和他的同事 (《Service- and Component-Based Development》, Addison-Wesley, 2004)、Heineman 和 Councill (《Component Based Software Engineering》, Addison-Wesley, 2001)、Brown (《Large-Scale Component-Based Development》, Prentice-Hall, 2000)、Allen (《Realizing e-Business with Components》, Addison-Wesley, 2000) 以及 Leavens 和 Sitaraman (《Foundations of Component-Based Systems》, Cambridge University Press, 2000) 编写的书覆盖了 CBSE 过程的许多重要方面。Stevens (《UML Components》, Addison-Wesley, 2006)、Apperly 和他的同事 (《Service- and Component-Based Development》, 2nd ed., Addison-Wesley, 2003) 以及 Cheesman 和 Daniels (《UML Components》, Addison-Wesley, 2000) 则侧重于用 UML 讨论 CBSE。

Malik (《Component-Based Software Development》, Lap Lambert Publishing, 2013) 提出了建立有效构件库的方法。Gross (《Component-Based Software Testing with UML》, Springer, 2010) 以及 Gao 和他的同事 (《Testing and Quality Assurance for Component-Based Software》, Artech House, 2006) 论述了基于构件的系统测试和 SQA 问题。

近些年出版了大量书籍来描述产业界基于构件的标准。这些著作既强调了关于制定标准本身的工作, 也讨论了许多重要的 CBSE 话题。

Linger、Mills 和 Witt 的著作 (《Structured Programming—Theory and Practice》, Addison-Wesley, 1979) 仍是设计方面的权威著作, 里面包含很好的 PDL, 以及关于结构化程序设计分支的细节讨论。其他书籍则集中在传统系统的过程设计问题方面, 如 Farrell (《A Guide to Programming Logic and Design》, Course Technology, 2010)、Robertson (《Simple Program Design》, 5th ed., Course Technology, 2006)、Bentley (《Programming Pearls》, 2nd ed., Addison-Wesley, 1999) 以及 Dahl (《Structured Programming》, Academic Press, 1997) 等。

最近出版的书籍中, 专门讨论构件级设计的书很少。一般来讲, 编程语言书籍或多或少关注于过程设计, 通常以书中所介绍的语言为环境进行讲解, 这方面有成百上千本书。

在网上有大量关于构件级设计的信息, 有关构件级设计的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 上找到。

用户界面设计

要点浏览

概念：用户界面（UI）设计在人与计算机之间搭建了一个有效的交流媒介。遵循一系列的界面设计原则，定义界面对象和界面动作，然后创建构成用户界面原型基础的屏幕布局。

人员：软件工程师通过迭代过程来设计用户界面，这个过程采纳了被广泛接受的设计原则。

重要性：不管软件展示了什么样的计算能力、发布了什么样的内容及提供了什么样的功能，如果软件不方便使用、常导致用户犯错或者不利于完成目标，你是不会喜欢这个软件的。由于界面影响用户对于软件的感觉，因此，它必须是令人满意的。

步骤：用户界面设计首先要识别用户、任

务和环境需求。一旦确定用户任务，则通过创建和分析用户场景来定义一组用户界面对象和动作。这是创建屏幕布局的基础。屏幕布局描述了图标的图形设计和位置、描述性屏幕文本的定义、窗口的规格说明和命名，以及主要的和次要的菜单项规格说明。可以使用工具来开发原型并最终实现设计模型，另外为了保证质量需要对结果进行评估。

工作产品：创建用户场景，构建产品屏幕布局，以迭代的方式开发和修改界面原型。

质量保证措施：原型的开发是通过用户测试驱动的，测试驱动的反馈将用于原型的下一次迭代修改。

我们生活在充满高科技产品的世界里。几乎所有这些产品，诸如消费电子产品、工业设备、汽车产品、企业系统、军事系统、个人计算机软件、移动 App 及 WebApp，都需要人参与交互。如果要使一个产品取得成功，它就必须展示出良好的可用性。可用性是指用户在使用高科技产品所提供的功能和特性时，对使用的容易程度和有效程度的定量测量。

在计算时代的前 30 年里，可用性并不是软件开发主要关心的。Donald Norman[Nor88] 在其关于设计的经典书籍中曾经主张（对待可用性的）态度改变的时机已经到来：

为了使技术适应人类，必须要研究人类。但我们现在倾向于只研究技术。结果，人们不得不顺从技术。而现在是时候扭转这个趋势，使技术适应人类了。

随着技术专家对人类交互的研究，出现了两个主要的问题。第一，定义一组黄金规则（15.1 节）。这些规则可以应用于所有与人交互的技术产品。第二，定义交互机制使软件设计人员建立起可以恰当实现黄金规则的

关键概念

- 可访问性
- 命令标记
- 控制
- 设计评估
- 错误处理
- 黄金规则
- 帮助设施
- 界面分析
- 界面一致性
- 界面设计
- 界面设计模型
- 国际化
- 记忆负担
- 原则和指导方针
- 过程

系统。这些机制消除了机器与人交互方面的一些难题，我们统一称之为用户界面。但即便是在今天，我们还是会遇到这样的用户界面：难学、难用、令人迷惑、不直观、不可原谅。在很多情况下，它们让人感到十分沮丧。然而，仍然有人在花费时间和精力去创建这样的界面，看起来，创建者并不是有意制造麻烦。

关键概念

响应时间
任务分析
任务细化
可用性
用户分析
WebApp 和移动
App 界面设计

317

15.1 黄金规则

Theo Mandel 在其关于界面设计的著作 [Man97] 中提出了三条黄金规则：

1. 把控制权交给用户。
2. 减轻用户的记忆负担。
3. 保持界面一致。

这些黄金规则实际上构成了一系列用户界面设计原则的基础，这些原则可以指导软件设计的重要方面。

15.1.1 把控制权交给用户

在重要的、新的信息系统的需求收集阶段，曾经征求一位关键用户对于窗口图形界面相关属性的意见。

该用户严肃地说：“我真正喜欢的是一个能够理解我想法的系统，它在我需要去做以前就知道我想做什么，并使我可以非常容易地完成。这就是我想要的，我也仅此一点要求。”

你的第一反应可能是摇头和微笑，但是，沉默了一会儿后，你会觉得该用户的想法绝对没有什么错。她想要一个对其要求能够做出反应并帮助她完成工作的系统。她希望去控制计算机，而不是计算机控制她。

设计者施加的大多数界面约束和限制都是为了简化交互模式。但是，这是为了谁呢？

在很多情况下，设计者为了简化界面的实现可能会引入约束和限制，其结果可能是界面易于构建，但会妨碍使用。Mandel[Man97] 定义了一组设计原则，允许用户掌握控制权。

以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式。交互模式就是界面的当前状态。例如，如果在字处理器菜单中选择拼写检查，则软件将转移到拼写检查模式。如果用户希望在这种情形下进行一些文本编辑，则没有理由强迫用户停留在拼写检查模式，用户应该能够几乎不需做任何动作就可以进入和退出该模式。

提供灵活的交互。由于不同的用户有不同的交互偏好，因此应该提供选择机会。例如，软件可能允许用户通过键盘命令、鼠标移动、数字笔、触摸屏或语音识别命令等方式进行交互。但是，每个动作并非要受控于每一种交互机制。例如，考虑使用键盘命令（或语音输入）来画一幅复杂形状的图形是有一定难度的。

允许用户交互被中断和撤销。即使陷入一系列动作之中，用户也应该能够中断动作序列去做某些其他事情（而不会失去已经做过的工作）。用户也应该能够“撤销”任何动作。

当技能水平高时可以使交互流线化并允许定制交互。用户经常发现他们重复地完成相同的交互序列，因此，值得设计一种“宏”机制，使得高级用户能够定制界面以方便交互。

引述 依照用户的习惯来设计，比纠正用户的习惯要好。

Jon Meads

318

引述 我一直希望计算机能像电话一样易于使用。我的希望已经实现了，我不再想知道如何使用电话了。

Bjarne Stronstrup

使用户与内部技术细节隔离开来。用户界面应该能够将用户移入应用的虚拟世界中，用户不应该知道操作系统、文件管理功能或其他隐秘的计算技术。

设计应允许用户与出现在屏幕上的对象直接交互。当用户能够操纵完成某任务所必需的对象，并且以一种该对象好像是真实存在的方式来操纵它时，用户就会有一种控制感。例如，允许用户将文件拖到“回收站”的应用界面，即是直接操纵的一种实现。

15.1.2 减轻用户的记忆负担

一个经过精心设计的用户界面不会加重用户的记忆负担，因为用户必须记住的东西越多，和系统交互时出错的可能性也就越大。只要可能，系统应该“记住”有关的信息，并通过有助于回忆的交互场景来帮助用户。Mandel[Man97]定义了一组设计原则，使得界面能够减轻用户的记忆负担。

减少对短期记忆的要求。当用户陷于复杂的任务时，短期记忆的要求会很强烈。界面的设计应该尽量不要要求记住过去的动作、输入和结果。可行的解决办法是通过提供可视的提示，使得用户能够识别过去的动作，而不是必须记住它们。

建立有意义的默认设置。初始的默认集合应该对一般的用户有意义，但是，用户应该能够说明个人的偏好。然而，“重置”（reset）选项应该是可用的，使得可以重新定义初始默认值。

定义直观的快捷方式。当使用助记符来完成系统功能时（如用 Alt+P 激活打印功能），助记符应该以容易记忆的方式联系到相关动作（例如，使用要激活任务的第一个字母）。

界面的视觉布局应该基于真实世界的象征。例如，一个账单支付系统应该使用支票簿和支票登记簿来指导用户的账单支付过程。这使得用户能够依赖于很好理解的可视化提示，而不是记住复杂难懂的交互序列。

以一种渐进的方式揭示信息。界面应该以层次化方式进行组织，即关于某任务、对象或行为的信息应该首先在高抽象层次上呈现。更多的细节应该在用户表明兴趣后再展示。

SafeHome 违反用户界面的黄金规则

[场景] Vinod 的工作间，用户界面设计启动在即。

[人物] Vinod 和 Jamie，SafeHome 软件工程团队成员。

[对话]

Jamie: 我已经在考虑监控功能的界面了。

Vinod (微笑): 思考是好事。

Jamie: 我认为我们可以将其简化。

Vinod: 什么意思？

Jamie: 如果我们完全忽略住宅平面图会怎么样？它倒是很华丽，但是会带来很多开发工作量。我们只要询问用户要查看的指定摄像机，然后在视频窗口显示视频就

可以了。

Vinod: 房主如何记住有多少个摄像机以及它们都安装在什么地方呢？

Jamie (有点不高兴): 他是房主，应该知道。

Vinod: 但是如果不知道呢？

Jamie: 应该知道。

Vinod: 这不是问题的关键……如果忘记了昵？

Jamie: 哦，我应该提供一张可操作的摄像机及其位置的清单。

Vinod: 那也有可能，但是为什么要有一份清单呢？

Jamie: 好的，无论用户是否有这方面的

要求，我们都提供一份清单。

Vinod：这样更好。至少用户不必特意记住我们给他的东西了。

Jamie（想了一会儿）：但是你喜欢住宅平面图，不是吗？

Vinod：哈哈。

Jamie：你认为市场营销人员会喜欢哪一个？

Vinod：你在开玩笑，是吗？

Jamie：不。

Vinod：哦……华丽的那个……他们喜欢迷人的……他们对简单的不感兴趣。

Jamie：（叹口气）好吧，也许我应该为两者都设计一个原型。

Vinod：好主意……我们就让客户来决定。

15.1.3 保持界面一致

用户应该以一致的方式展示和获取信息，这意味着：（1）按照贯穿所有屏幕显示的设计规则来组织可视信息；（2）将输入机制约束到有限的集合，在整个应用中得到一致的使用；（3）从任务到任务的导航机制要一致地定义和实现。Mandel[Man97]定义了一组帮助保持界面一致性的设计原则。

允许用户将当前任务放入有意义的环境中。很多界面使用数十个屏幕图像来实现复杂的交互层次。提供指示器（例如，窗口标题、图标、一致的颜色编码）帮助用户知晓当前工作环境是十分重要的。另外，用户应该能够确定他来自何处以及存在哪些转换到新任务的途径。

在完整的产品线内保持一致性。一个应用系列（即一个产品线）都应采用相同的设计规则，以保持所有交互的一致性。

如果过去的交互模型已经建立起了用户期望，除非有不得已的理由，否则不要改变它。一个特殊的交互序列一旦变成事实上的标准（如使用 Alt+S 来存储文件），则用户在遇到每个应用时均会如此期望。如果改变这些标准（如使用 Alt+S 来激活缩放比例），将导致混淆。

本节和前面几节讨论的界面设计原则为软件工程师提供了基本指南。在下面几节中，我们将考察界面设计过程。

引述 看起来不同的事物产生的效果应该不同，而看起来相同的事物产生的效果应该相同。

Larry Marine

321

信息栏

可用性

在一篇关于可用性的见解深刻的论文中，Larry Constantine[Con95]提出了一个与可用性主题非常相关的问题：“用户究竟想要什么？”他给出了下面的回答。

“用户真正想要的是好的工具。所有的软件系统，从操作系统和语言到数据录入和决策支撑应用软件，都是工具。最终用户希望从为其设计的工具中得到的东西，与我们希望从所使用工具中得到的是

一样的。他们想要易于学习并能够为自己的工作提供帮助。同时，他们想要的系统应该能提高工作效率，不会欺骗或困扰他们，不会使他们易于犯错误或难于完成工作。”

Constantine 指出，系统的可用性并非取决于设计美学、交互技术的发展水平或者内置的界面智能等方面，而是当界面的架构适合于将要使用这些界面的用户的需求

求时,才能获得可用性。

正式的可用性定义往往令人有些迷惑。Donahue 和他的同事 [Don99] 给出了如下的定义:“可用性是一种衡量计算机系统好坏的度量……便于学习;帮助初学者记住他们已经学到的东西;降低犯错的可能;使得用户更加有效率,并且使得他们对系统感到满意。”

确定你所建系统是否可用的唯一办法就是进行可用性评估和测试。观察用户与系统的交互,同时回答下列问题 [Con95]:

- 在没有连续的帮助或用法说明的情况下,系统是否便于使用?
- 交互规则是否能够帮助一个知识渊博的用户工作得更加有效率?
- 随着用户的知识不断增多,交互机制是否能变得更灵活?
- 系统是否已经过调试,使之适应其运行的物理环境和社会环境?

- 用户是否意识到系统的状态?在工作期间,用户是否能够知道其所处的位置?
- 界面是否是按照一种合理并且一致的方式来构建的?
- 交互机制、图标和过程是否在整个界面中一致?
- 交互是否能够提前发现错误并帮助用户修正它们?
- 界面是否能够容错?
- 交互是否简单?

如果上述每个问题的回答都是肯定的,那么可以认为这个系统是可用的。

可用性好的系统带来的诸多好处在于 [Don99]: 提高销售量和用户满意度、具有竞争优势、在媒体中获得良好的评价、获得良好的口碑、降低支持成本、提升最终用户生产力、降低培训费用、减少文档开销、减少来自不满意用户的投诉。

15.2 用户界面的分析和设计

用户界面的分析和设计全过程始于创建不同的系统功能模型(从外部看时对系统的感觉)。首先将完成系统功能的任务分为面向人的和面向计算机的,然后考虑那些应用到界面设计中的各种设计问题。可以使用各种工具来建造原型并最终实现设计模型,最后由最终用户从质量的角度对结果进行评估。

网络资源 可以在 www.nngroup.com 找到用户界面设计信息的优秀资源。

15.2.1 用户界面分析和设计模型

分析和设计用户界面时要考虑四种模型:工程师(或者软件工程师)建立用户模型;软件工程师创建设计模型;最终用户在脑海里对界面产生映像,称为用户的心理模型或系统感觉;系统的实现者创建实现模型。不幸的是,这四种模型可能相差甚远。界面设计人员的任务就是消解这些差距,导出一致的界面表示。

用户模型确立了系统最终用户的轮廓(profile)。Jeff Patton [Pat07] 在《用户为中心的设计》(user-centric design)的前言中写道:

事实是,设计者和开发者(包括我自己)都经常考虑到用户。然而,在缺少特定用户有力的心理模型的情况下,开发者和设计者会以自我来替代用户。自我替代并不是用户为中心,而是自我为中心。

引述 如果用户界面有一点瑕疵,那么整个用户界面都会被破坏。

Douglas
Anderson

为了建立有效的用户界面，“开始设计之前，必须对预期用户加以了解，包括年龄、性别、身体状况、教育、文化和种族背景、动机、目标以及性格”[Shn04]。此外，可以将用户分类：新手，对系统有了解的用户，间歇用户或对系统有了解的经常用户。

用户的心理模型（系统感觉）是最终用户在脑海里对系统产生的印象，例如，请某个餐厅评级移动 App 的用户来描述其操作，那么系统感觉将会引导用户的回答，准确的回答取决于用户的经验（新手只能做简要的回答）和用户对应应用领域软件的熟悉程度。一个对餐厅评级应用程序有深刻了解但只使用这种系统几次的用户，可能比已经使用该系统好几个星期的新手对该应用程序的功能描述回答得更详细。

实现模型组合了计算机系统的外在表现（界面的观感），结合了所有用来描述系统语法和语义的支撑信息（书、手册、录像带、帮助文件）。当系统实现模型和用户心理模型相一致的时候，用户通常就会对软件感到很舒服，使用起来就很有效。为了将这些模型融合起来，所开发的设计模型必须包含用户模型中的一些信息，实现模型必须准确地反映界面的语法和语义信息。

建议 即使用户是新手也会有使用快捷键的需求；即使是经常使用系统的用户有时候也需要指导。他们的要求都要满足。

引述 注意用户的行为而不是他们的言语。

Jakob Nielsen

15.2.2 过程

用户界面的分析和设计过程是迭代的，可以用类似于第 4 章讨论过的螺旋模型表示。如图 15-1 所示，用户界面分析和设计过程开始于螺旋模型的内部，且包括四个不同的框架活动 [Man97]：（1）界面分析和建模；（2）界面设计；（3）界面构建；（4）界面确认。图 15-1 中的螺旋意味着每个活动都将多次出现，每绕螺旋一周表示需求和设计的进一步细化。在大多数情况下，构建活动涉及原型开发——这是唯一实用的确认设计结果的方式。

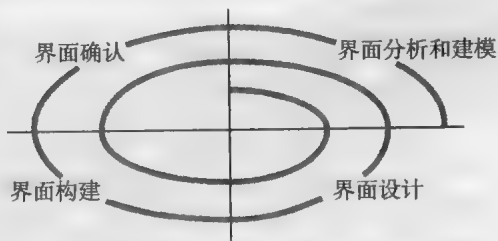


图 15-1 用户界面的设计过程

界面分析活动的重点在于那些与系统交互的用户的轮廓。记录技能级别、业务理解以及对新系统的一般感悟，并定义不同的用户类别。对每个用户类别进行需求引导。本质上，软件工程师试图去理解每类用户的系统感觉（15.2.1 节）。

一旦定义好了一般需求，就将进行更详细的任务分析。标识、描述和细化（通过绕螺旋的多次迭代）用户为了达到系统目标而执行的任务。15.3 节将对任务分析进行更详细的讨论。最后，用户环境的分析着重于物理工作环境的特征（例如地理位置、采光、位置约束）。

作为分析动作的一部分而收集的信息被用于创建界面的分析模型。使用该模型作为基础，设计活动便开始了。

界面设计的目标是定义一组界面对象和动作（以及它们的屏幕表示），使得用户能够以满足系统所定义的每个使用目标的方式完成所有定义的任务。界面设计将在 15.4 节详细讨论。

界面构建通常开始于创建可评估使用场景的原型。随着迭代设计过程的继续，用户界面开发工具（15.5 节）可用来完成界面的构造。

界面确认着重于：（1）界面正确地实现每个用户任务的能力，适应所有任务变化的能力以及达到所有一般用户需求的能力；（2）界面容易使用和学习的程度；（3）作为工作中的得

力工具，用户对界面的接受程度。

如我们已经提到的，本节描述的活动是以迭代方式开展的。因此，不需要在第一轮就试图刻画所有的细节（对分析或设计模型而言）。后续的过程将细化界面的任务细节、设计信息和运行特征。

15.3 界面分析^①

所有软件工程过程模型的一个重要原则是：在试图设计一个解决方案之前，最好对问题有所理解。在用户界面的设计中，理解问题就意味着了解：（1）通过界面和系统交互的人（最终用户）；（2）最终用户为完成工作要执行的任务；（3）作为界面的一部分而显示的内容；（4）任务处理的环境。在接下来的几节中，为了给设计任务建立牢固的基础，我们来检查界面分析的每个成分。

15.3.1 用户分析

在担忧技术上的问题之前，用户界面这个词完全有理由要求我们花时间去理解用户。之前，我们提到每个用户对于软件都存在心理映像，而这可能与其他用户的心理映像存在着差别。另外，用户的心理映像可能与软件工程师的设计模型相距甚远。设计师能够将得到的心理映像和设计模型聚合在一起的唯一办法就是努力了解用户，同时了解这些用户是如何使用系统的。为了完成这个任务，可以利用各种途径（用户访谈、销售输入、市场输入、支持输入）获得的信息。

下列一组问题（改编自 [Hac98]）将有助于界面设计师更好地理解系统的用户：

- 用户是经过训练的专业人员、技术员、办事员，还是制造业工人？
- 用户平均正规教育水平如何？
- 用户是否具有学习书面资料的能力或者是否渴望接受集中培训？
- 用户是专业录入人员还是键盘恐惧者？
- 用户群体的年龄范围如何？
- 是否需要考虑用户的性别差异？
- 如何为用户完成的工作提供报酬？
- 用户是否在正常的办公时间内工作或者一直干到工作完成？
- 软件是用户所完成工作中的一个集成部分，还是偶尔使用一次？
- 用户群中使用的主要交流语言是什么？
- 如果用户在使用软件的过程中出错，结果会怎么样？
- 用户是否是系统所解决问题领域的专家？
- 用户是否想了解界面背后的技术？

提问 我们如何知道最终用户的人数和特征？

这些问题和类似问题的答案将帮助设计师了解：最终用户是什么人，什么可能令他们感到愉悦，如何对用户进行分类，他们对系统的心理模型是什么样子，用户界面必须具有哪些特性才能满足用户的需求。

^① 因为需求分析问题在第 8~11 章已经讨论过，所以有理由把这一节放到这几章中去。本节之所以放在这里，是因为界面的分析和设计紧紧相连，两者的界限常常模糊不清。

15.3.2 任务分析和建模

任务分析的目标就是给出下列问题的答案：

- 在指定环境下用户将完成什么工作？
- 用户工作时将完成什么任务和子任务？
- 在工作中用户将处理什么特殊的问题域对象？
- 工作任务的顺序（工作流）如何？
- 任务的层次关系如何？

为了回答这些问题，软件工程师必须利用本书前面所讨论的分析技术，只不过在此种情况下，要将这些技术应用到用户界面。

用例。在前面几章中，我们提到过用例描述了参与者（在用户界面设计中，参与者通常是某个人）和系统的交互方式。作为任务分析的一部分，设计用例用来显示最终用户如何完成指定的相关工作任务。在大多数情况下，用例采用第一人称并以非正式形式（一段简单的文字）来书写。例如，假如一家小的软件公司想专门为公司室内设计师开发一个计算机辅助设计系统。为了更好地理解他们是如何工作的，实际的室内设计师应该描述特定的设计功能。在室内设计师被问到“如何确定室内家具摆放位置”的时候，室内设计师写下了如下非正式的用例描述：

我从勾画房间的平面图、窗户与门的尺寸和位置开始设计。我非常关心射入房间的光线，关心窗外的风景（如果它很漂亮，就会吸引我的注意力），关心无障碍墙的长度，关心房间内活动空间的通道大小。我接下来会查看客户和我选取的家具清单……接着，我会为客户画出一个房屋的透视图（三维图画），让客户感受到房间看起来应该是什么样的。

这个用例给出了计算机辅助设计系统中一项重要工作任务的基本描述。从这个描述中，软件工程师能够提炼出任务、对象和整个交互流程。另外，系统中能够使得室内设计师感到愉悦的其他特征也被构思出来。例如，可以将房屋中每一扇窗户的风景都拍摄成一张数码相片。在画房屋透视图时，通过每扇窗户就可以看到窗外的真实景象。

关键点 用户的目的是通过用户界面来完成一个或多个任务。为了实现这一点，用户界面必须提供用户达到目标的机制。

网络资源 可以在 <http://web.eecs.umich.edu/~kieras/docs/GOMS/> 找到不错的用户建模信息资源。

326

SafeHome 用户界面设计的用例

[场景] Vinod 的工作间，用户界面设计正在进行。

[人物] Vinod 和 Jamie，SafeHome 软件工程团队成员。

[对话]

Jamie：我拦住我们的市场部联系人，让她写了一份监视界面的用例。

Vinod：站在谁的角度来写？

Jamie：当然是房主，还会有谁？

Vinod：还有系统管理员这个角色。即使是房主担任这个角色，这也是一个不同的视角。“管理员”启动系统，配置零件，

布置平面图，安置摄像机……

Jamie：当房主想看视频时，我只是让她扮演房主的角色。

Vinod：好的，这只是监视功能界面主要行为之一。但是，我们也应该调查一下系统管理员的行为。

Jamie（有些不满）：你是对的。

（Jamie 离开去找销售人员。几个小时以后她回来了。）

Jamie：我真走运，找到了市场部联系人，我们一起完成了系统管理员的用例。我们应该把“管理”定义为可以应用所有其他

SafeHome 功能的一个功能。这是我们提出的用例。

(Jamie 给 Vinod 看这个非正式的用例。)

非正式用例：我想能够在任何时候设置和编辑系统的布置方案。当我启动系统时，我选择某个管理功能。系统询问我是否要建立一个新的系统布置方案，或者询问我是否编辑已有的方案。如果我选择了一个新建方案，系统呈现一个绘画屏幕，在网格上可以画出建筑平面图来。为了绘画简便，应该提供墙壁、窗户和门的图标。我只是将图标伸展到合适的长度。系统将把长度显示为英尺或者米（我可以选择度量系统）。我能够从传感器和摄像机库中进行选择，并且将它们放置在平面图中。我

标记每个传感器和摄像机，或者系统自动进行标记。我可以通过合适的菜单对传感器和摄像机进行设置。如果选择编辑，就可以移动传感器和摄像机，添加新的或删除已有的传感器和摄像机，编辑平面图并编辑摄像机和传感器的设置。在每种情形下，我希望系统能够进行一致性检查并且帮助我避免出错。

Vinod (看完脚本之后)：好的，对于绘画程序，可能有一些有用的设计模式（第 16 章）或可复用的图形用户界面构件。我打赌，通过使用可复用构件，我们可以实现某些或大部分管理员界面。

Jamie：同意！我马上进行检查。

任务细化。在第 12 章中，我们讨论了逐步求精（也称为功能分解或者逐步细化），把它作为一种细化处理任务的机制，而这些任务是软件完成某些期望功能所要求的。界面设计的任务分析采用了一种详细阐述的办法来辅助理解用户界面必须采纳的用户活动。

327

首先，工程师必须定义完成系统或应用程序目标所需的任务并对任务进行划分。例如，考虑前面讨论的为室内设计师开发的计算机辅助设计系统。通过观察工作中的室内设计师，软件工程师了解到，室内设计由一系列的主要活动组成：家具布置（在前面用例设计中提到过）、结构和材料的选择、墙壁和窗户装饰物的选择、（向客户）展示、计算成本、购物。其中任何一个都可以被细化成一系列的子任务。例如，使用用例中的信息，可以将家具布置任务细化为下面的子任务：（1）根据房屋的尺寸画出平面图；（2）将门窗安置在合适的位置；（3a）使用家具模型在平面图上描绘相应比例的家具轮廓；（3b）使用饰件模板在平面设计图上勾勒相应比例的饰件；（4）移动家具和饰件轮廓线到达理想的位置；（5）标记所有的家具和饰件轮廓；（6）标出尺寸以显示其位置；（7）为用户勾画透视图。也可以应用类似的方法对其他主任务进行细化。

上面的每个子任务都可以进一步细化。其中子任务 1～6 可以通过在界面中操纵信息和执行各种动作来完成。另一方面，子任务 7 可以在软件中自动完成，并且几乎不用直接与用户交互^①。界面的设计模型应该以一种与用户模型（典型室内设计师的轮廓图）和系统感觉（室内设计师期望系统自动提供）相一致的方式来配合这些任务。

对象细化。软件工程师这时不是着眼于用户必须完成的任务，而是需要检查用例和来自用户的其他信息，并且提取室内设计师需要使用的物理对象。这些对象可以分为不同的类。需要定义每个类的属性，并且通过对

建议 尽管对象细化十分有用，但它应当作为独立的方法去使用。任务分析的过程中，应当考虑用户的声音。

① 然而，事实可能不是这样。室内设计师可能想要指定所画的透视图、缩放比例、色彩的运用和其他信息。与透视渲染相关的用例将提供解决这些问题的信息。

每个对象动作的评估为设计师提供一个操作列表。例如，家具模板可能被转换成一个名为 Furniture 的类，这个类包括 size、shape 和 location 等属性。室内设计师会从 Furniture 类中选择对象，将其移动到平面图（在此处，平面图是另一个对象）中的某个位置上，拖曳家具的轮廓，依此类推。任务选择（select）、移动（move）、拖曳（draw）等都是操作。用户界面分析模型不能对任何一种操作都提供文字实现。然而，随着设计的不断细化，对每个操作的细节都会进行定义。

workflow 分析。当大量扮演着不同角色的用户使用某个用户界面时，有时候除了任务分析和对象细化之外，还有必要进行 workflow 分析。该技术使得软件工程师可以很好地理解在涉及多个成员（角色）时，工作过程是如何完成的。假设某个公司打算将处方药的开方和给药过程全部自动化。全部过程^①将围绕着一个 WebApp 进行考虑，医生（或者他们的助手）、药剂师和病人等都可以访问这个应用系统。用 UML 泳道图（活动图的一种变形）能够有效地表示 workflow。

[328]

下面只考虑工作过程中的一小部分：当病人请求重填处方时发生的情形。图 15-2 给出了一个泳道图，该图表明了前面提及的三个角色的任务和决定。这些信息可以通过访谈或每个角色书写的用例获取。不管怎样，事件流（图中显示的）使得界面设计师认识到三个关键的界面特征：

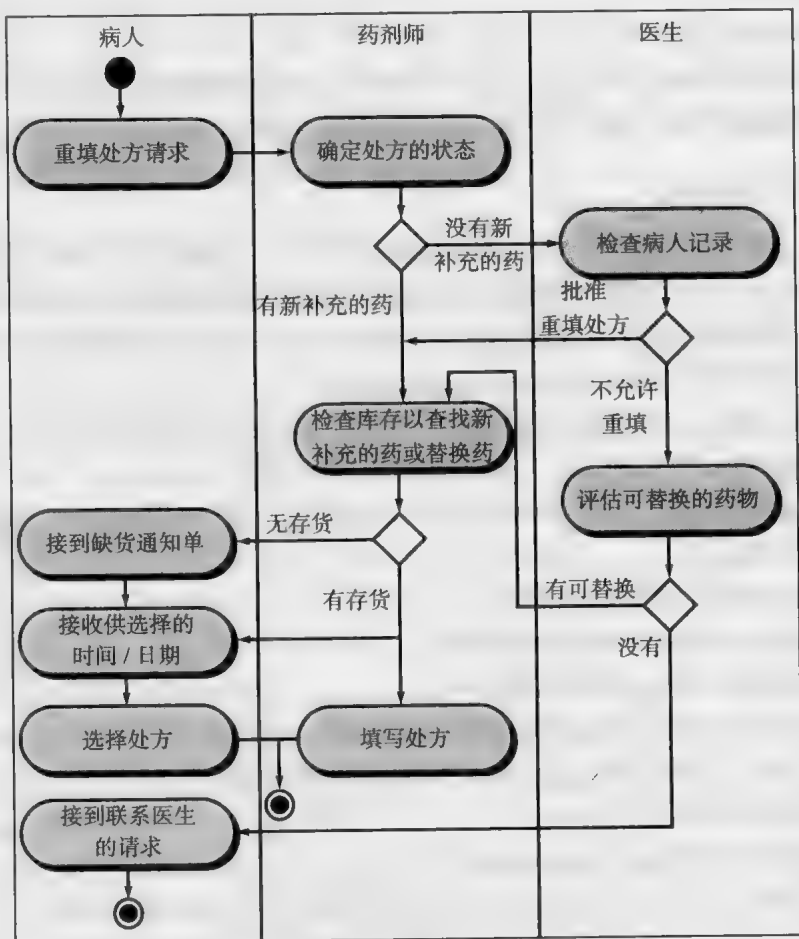


图 15-2 处方重填功能的泳道图

① 这个例子选自 [Hac98]。

1. 每个用户通过界面实现不同的任务，因此，为病人设计的界面在感官上与为药剂师或医生设计的界面有所不同。
2. 为医生和药剂师设计的界面应该能够访问和显示来自辅助信息源的信息（例如，药剂师应能够访问库存详细清单，而医生应能够访问其他可选药物信息）。
3. 泳道图中的很多活动都可以采用任务分析和对象求精使其进一步细化（例如，“填写处方”隐含着邮购支付、访问药房，或者访问特殊药品分发中心）。

引述 使技术适应用户要比用户适应技术好。

Larry Marine

层次表示。在界面分析时，会产生相应的细化过程。一旦建立了工作流，就为每个用户类型都定义一个任务层次。该任务层次来自于为用户定义的每项任务的逐步细化。例如，考虑重填处方的用户任务请求，任务层次如下：

重填处方请求

- 提供辨识信息
 - 指定姓名
 - 指定用户 ID
 - 指定个人身份识别号码 (PIN) 和密码
- 指定处方序号
- 指定重填处方所需的日期

为了完成填写处方的任务，定义了三个子任务。可以将其中的第一个子任务“提供辨识信息”进一步细化成三个另外的子任务。

15.3.3 显示内容分析

15.3.2 节中标识出的用户任务导致需要对各种各样不同类型的内容进行描述。在第 9 章和第 11 章中讨论过的分析建模技术标识出由应用产生的输出数据对象。这些数据对象可能：（1）由应用中其他部分的构件（与界面无关）生成；（2）由应用所访问数据库中存储的数据获得；（3）从系统外部传递到正在讨论的应用中。

在界面分析步骤中，要考虑内容的格式和美感（当它要显示在界面上时）。其中需要提问和回答的问题包括：

- 不同类型的数据是否要放置到屏幕上固定的位置（例如，照片一般显示在右上角）？
- 用户能否定制内容的屏幕位置？
- 是否对所有内容赋予适当的屏幕标识？
- 为了便于理解，应如何划分长篇报告？
- 对于大集合的数据，是否存在直接移动到摘要信息的机制？
- 输出图形的大小是否需要适合所使用显示设备的限制？
- 如何使用颜色来增强理解？
- 出错信息和警告应如何呈现给用户？

提问 作为用户界面设计的一部分，我们如何决定所显示内容的格式和美感？

对这些（和其他）问题的回答有助于软件工程师建立起内容表示的需求。

15.3.4 工作环境分析

Hackos 和 Redish[Hac98] 在讨论工作环境分析的重要性时这样写道：“人们不能孤立地

完成任务。他们会受到周围活动的影响,如工作场所的物理特征,使用设备的类型,与其他人的工作关系等。”在某些应用中,计算机系统的用户界面被放在“用户友好”的位置(例如,合适的亮度、良好的显示高度、简单方便的键盘操作),但有些地方(例如,工厂的地板和飞机座舱)亮度可能不是很适合,噪音也可能是个问题,也许不能选择使用键盘、鼠标或触摸屏,显示方位也不甚理想。界面设计师可能会受到某些因素的限制,这些因素会减弱易用性。

除了物理的环境因素之外,工作场所的文化氛围也起着作用。可否采用某种方式(例如,每次交互所用时间、交互的准确性)来度量系统的交互?在提供一个输入前,两个或多个人员是否一定要共享信息?如何为系统用户提供支持?在界面设计开始之前,应该对上述问题和更多的相关问题给予回答。

331

15.4 界面设计步骤

一旦完成了界面分析,最终用户要求的所有任务(对象和动作)都已经被详细确定下来,界面设计活动就开始了。与所有的软件工程设计一样,界面设计是一个迭代的过程。每个用户界面设计步骤都要进行很多次,每次精细化的信息都来源于前面的步骤。

引述 交互设计是图形艺术、技术和心理学的无缝结合。

Brad Wieners

尽管已经提出了很多不同的用户界面设计模型(例如[Nor86]和[Nie00]),但它们都建议结合以下步骤:(1)定义界面对象和动作(操作);(2)确定事件(用户动作),即会导致用户界面状态发生变化的事件;(3)描述每个状态的表示形式;(4)说明用户如何利用界面提供的信息来解释每个状态。

15.4.1 应用界面设计步骤

界面设计的一个重要步骤是定义界面对象和作用于对象上的动作。为了完成这个目标,需要使用类似于第9章介绍的方法来分析用户场景,也就是说,撰写用例的描述。名词(对象)和动词(动作)被分离出来形成对象和动作列表。

一旦完成了对象和动作的定义及迭代细化,就可以将它们按类型分类。目标、源和应用对象都被标识出来。将源对象(如报告图标)拖放到目标对象(如打印机图标)上,这意味着该动作要产生一个硬拷贝的报告。应用对象代表应用中特有的数据,它们并不作为屏幕交互的一部分被直接操纵。例如,邮件列表被用于存放邮件的名字,该列表本身可以进行排序、合并或清除(基于菜单的动作),但是,它不会通过用户的交互被拖动和删除。

当设计者满意地认为已经定义了所有的重要对象和动作(对一次设计迭代而言)时,便可以开始进行屏幕布局。与其他界面设计活动一样,屏幕布局是一个交互过程,其中包括:图标的图形设计和放置、屏幕描述性文字的定义、窗口的规格说明和标题,以及各类主要和次要菜单项的定义等。如果一个真实世界的隐喻适合于该应用,则在此时进行说明,并以补充隐喻的方式来组织布局。

为了对上面的设计步骤提供简明的例证,我们考虑 SafeHome 系统(在前面几章讨论过)的一个用户场景。下面是界面的初步用例(由房主写的)描述。

初步用例:我希望通过 Internet 在任意的远程位置都能够访问 SafeHome 系统。使用运行在笔记本上的浏览器软件(当正处于工作或者旅行状态时),我可以决定报警系统的状态、启动或关闭系统、重新配置安全区以及通过预先安置的摄像机观察住宅内的不同房间。

332

为了远程访问 SafeHome，我需要提供标识符和密码，这些定义了访问的级别（如并非所有用户都可以重新配置系统）并提供安全保证。一旦确认了身份，我就可以检查系统状态，并通过启动或关闭 SafeHome 系统改变状态。通过显示住宅的平面图，观察每个安全传感器，显示每个当前配置区域以及修改区域（必要时），可以重新配置系统。通过有策略地放置摄像机以观察房子内部。通过对每个摄像机进行摇动和变焦以提供房子内部的不同视角。

基于这个用例，确定房主的任务、对象和数据项如下：

- 访问 SafeHome 系统。
- 输入 ID 和密码实现远程访问。
- 检查系统状态。
- 启动或关闭 SafeHome 系统。
- 显示平面图和传感器位置。
- 显示平面图上的区域。
- 改变平面图上的区域。
- 显示建筑平面图上的视频摄像机位置。
- 选择用于观察的视频摄像机。
- 观察视频图像（每秒 4 帧）。
- 摇动或变焦摄像机。

从房主的这个任务清单中抽取出对象和动作。所提到的大部分对象都是应用对象。然而，视频摄像机位置（源对象）被拖放到视频摄像机（目标对象）以创建视频图像（视频显示的窗口）。

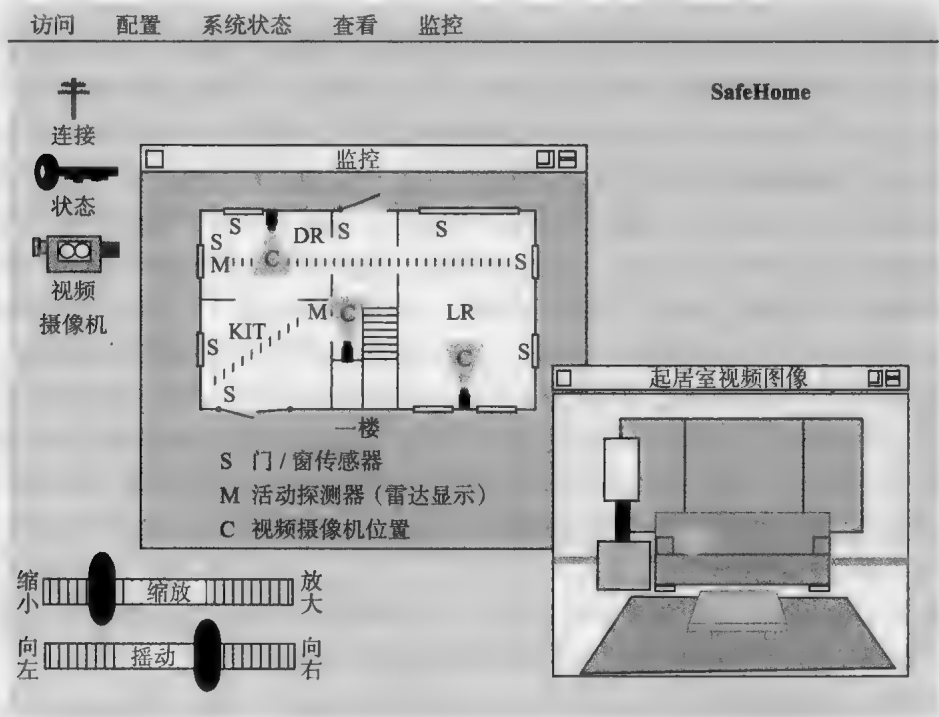


图 15-3 基本的屏幕布局

为视频监控设计的屏幕布局初步草图如图 15-3 所示^①。为了调用视频图像,需选择显示在监控窗口中的建筑平面图上的视频摄像机位置图标 C。在这种情况下,起居室(LR)中的摄像机位置被拖放到屏幕左上部分的视频摄像机图标处,此时,视频图像窗口出现,显示来自位于起居室中的摄像机的流视频。变焦和摇动控制条用于控制视频图像的放大和方向。为了选择来自另一个摄像机的图像,用户只需简单地将另一个不同的摄像机位置图标拖放到屏幕左上区域的摄像机图标上即可。

所显示的布局草图需以菜单条上每个菜单项的扩展来补充,指明视频监控模式(状态)有哪些可用的动作。在界面设计过程中,将创建用户场景中提到的房主的每个任务的一组完整草图。

建议 尽管自动化的工具在开发布局原型中十分有用,但有时候铅笔和纸张也是需要的。

15.4.2 用户界面设计模式

图形用户界面已经变得如此普遍,以至于涌现出各式各样的用户界面设计模式。设计模式是一种抽象,描述了特定的、界限明确的设计问题的解决方案。

作为通常碰到的界面设计问题的一个例子,考虑用户必须一次或多次输入日历日期这种情况,有时候需要提前输入月份。对于这个简单的问题,有很多可能的解决方案,为此也提出了很多种不同的模式。Laakso[Laa00]提出了一种称为 CalendarStrip 的模式,此模式生成一个连续、滚动的日历,在这个日历上,当前日期被高亮度显示,未来的日期可以在日历上选择。这个日历隐喻在用户中具有很高的知名度,并提供了一种有效的机制,可以在上下文中设置未来的日期。

在过去的十年间,人们已经提出了很多用户界面设计模式。在第 16 章中有关于用户界面设计模式的更为详尽的论述。此外,Ericksen[Eri08]提供了许多基于 Web 的文献资料。

15.4.3 设计问题

在进行用户界面设计时,几乎总会遇到以下四个问题:系统响应时间、用户帮助设施、错误信息处理和命令标记。不幸的是,许多设计人员往往很晚才注意到这些问题(有时在操作原型已经建立起来后才发现问题),这往往会导致不必要的反复、项目拖延及用户的挫折感,最好的办法是在设计的初期就将这些作为设计问题加以考虑,因为此时修改比较容易,代价也低。

响应时间。系统响应时间包括两个重要的属性:时间长度和可变性。如果系统响应时间过长,用户就会感到焦虑和沮丧。系统时间的可变性是指相对于平均响应时间的偏差,在很多情况下这是最重要的响应时间特性。即使响应时间比较长,响应时间的低可变性也有助于用户建立稳定的交互节奏。例如,稳定在 1 秒的命令响应时间比从 0.1 秒到 2.5 秒不定的响应时间要好。在可变性到达一定值时,用户往往比较敏感,他们总是关心界面背后是否发生了异常。

帮助设施。几乎所有计算机交互式系统的用户都时常需要帮助。现代

网络资源 已经有大量的用户界面设计模式,访问 <http://www.hci-patterns.org/patterns/borchers/patternindex.html> 可以找到它们的站点链接。

引述 当试图设计一些十分简单的东西时,人们经常犯的共性错误就是低估了笨人的智慧。

Douglas Adams

① 注意这里的实现与前几章讲到的这些特性的实现有所不同。这里应该是第一次设计的草图,可以考虑提供备选的设计草图。

的软件均提供联机帮助，用户可以不离开用户界面就解决问题。

错误处理。通常，交互式系统给出的出错消息和警告应具备以下特征：（1）以用户可以理解的语言描述问题；（2）应提供如何从错误中恢复的建设性意见；（3）应指出错误可能导致哪些不良后果（比如破坏数据文件），以使用户检查是否出现了这些情况（或者在已经出现的情况下进行改正）；应伴随着视觉或听觉上的提示，并且永远不应该把错误归咎于用户。

[335]

菜单和命令标记。键入命令曾经是用户和系统交互的主要方式，并广泛用于各种应用。现在，面向窗口的界面采用点击（point）和选取（pick）方式，减少了用户对键入命令的依赖。但许多高级用户仍然喜欢面向命令的交互方式。在提供命令或菜单标签交互方式时，必须考虑以下问题：

- 每个菜单选项是否都有对应的命令？
- 以何种方式提供命令？有三种选择：控制序列（如 Alt+P）、功能键或键入命令。
- 学习和记忆命令的难度有多大？命令忘了怎么办？
- 用户是否可以定制和缩写命令？
- 在界面环境中菜单标签是否是自解释的？
- 子菜单是否与主菜单项所指功能相一致？
- 有适合于应用系列内部的命令使用约定吗？

应用的可访问性。随着计算型应用变得无处不在，软件工程师必须确保界面设计中包含使得有特殊要求的用户易于访问的机制。对于那些实际上面临挑战的用户（和软件工程师）来说，由于道义、法律和业务等方面的原因，可访问性是必需的。有多种可访问性指导方针（如 [W3C03]）——很多都是为 WebApp 设计的，但这些方针经常也能应用于所有软件——为设计界面提供了详细的建议，以使界面能够达到各种级别的可访问性。其他指南（如 [App13]、[Mic13]）对于“辅助技术”提供了专门的指导，这些技术用来解决那些在视觉、听觉、活动性、语音和学习等方面有障碍的人员的需要。

国际化。软件工程师和他们的经理往往会低估建立一个适应不同国家和不同语言需要的用户界面所应付出的努力和技能。用户界面经常是为一个国家和一种语言所设计的，在面对其他国家时只好应急对付。设计师面临的挑战就是设计出“全球化”的软件。也就是说，用户界面应该被设计成能够容纳需要交付给所有软件用户的核心功能。本地化特征使得界面能够针对特定的市场进行定制。

软件工程师有多种国际化指导方针（如 [IBM13]）可以使用。这些方针解决了宽度设计问题（例如，在不同的市场情况下屏幕布局可能是不同的），以及离散实现问题（例如，不同的字母表可能生成特定的标识和间距需求）。对于几十种具有成百上千字母和字符的自然语言的管理，已经提出的 Unicode 标准 [Uni03] 就是用来解决这个挑战性问题的。

[336]

引述 来自地狱的界面——修正这个错误并且继续进行，请输入任一个 11 位的素数……

作者不详

网络资源 开发可访问软件的指导原则可以在 <http://www-03.ibm.com/able/guidelines/software/access-software.html> 找到。

软件工具 用户界面开发

[目标] 用户界面开发工具使得软件工程师只需做有限的定制开发就可以建立复杂的

图形用户界面。这些工具提供了对可复用构件的访问，并且通过选择工具上预定义

的功能就可以建立用户界面。

[机制] 现代用户界面由一组可复用的构件组成, 这些构件与一些提供特殊特性的定制构件相结合。大多数用户界面的开发工具能够通过使用“拖放”功能来完成界面的设计。换句话说, 开发人员选择预定义的功能(例如, 表格构造器、交互机制、命令处理), 并将这些功能放置在所创建界面的环境中。

[代表性工具]^①

- LegaSuite GUI。由 Seagull Software (<http://www-304.ibm.com/partnerworld/gsd/solutiondetails.do?solution=1020&expand=true&lc=en>) 开发, 能够创建基于浏览器的

的图形用户界面 (GUI) 并且提供了对过时界面的再造功能。

- Motif Common Desktop Environment。由 Open Group (www.osf.org/tech/desktop/cde/) 开发, 是一个集成的图形用户界面, 用于开放系统桌面计算。它对数据、文件(图形化桌面)和应用系统的管理提供了单一的、标准的图形化界面。
- Alita Design 8.0。由 Altia (www.altia.com) 开发, 是一种可以在多种平台(例如, 自动的、手持的、工业的)上创建图形用户界面 (GUI) 的工具。

15.5 WebApp 和移动 App 的界面设计

无论是为 WebApp、移动设备、传统的软件应用、消费产品设计的用户界面, 还是为工业设备设计的用户界面, 都应该展示出本章前面所讲的特性。Dix[Dix99] 认为 Web 工程师设计的 WebApp 和移动界面必须能够回答用户三个主要问题: 我在哪里? 我现在能做什么? 我去过哪里? 我能够去哪里? 这些问题的答案使用户理解交互环境并且使应用更为有效。

引述 如果一个站点非常好用, 但却缺少美观、合适的设计风格, 同样会失败。

Curt Cloninger

15.5.1 界面设计原则与指导方针

WebApp 或移动 App 的用户界面是它的“第一印象”。不管它的内容、处理能力、服务以及应用自身的整体效益如何, 一个设计糟糕的用户界面将会使潜在的用户失去信心。事实上, 用户甚至可能转向使用别的应用, 因为几乎在每个主题领域内, WebApp 和移动 App 的竞争都是十分激烈的, 用户界面应当迅速“抓住”潜在用户。

当然, WebApp 和移动 App 之间具有重要的差异。由于小型移动设备(如智能手机)的物理限制, 移动界面设计师必须以集中的方式来压缩交互。然而, 本节讨论的基本原则仍然适用。

Bruce Tognozzi [Tog01] 定义了一组可用性更高的基本的设计原则。^②

预测。应用应当能够预测出用户的下一个动作。例如, 假设用户已经请求了一个内容对象, 此对象显示出针对最新版本操作系统的打印机驱动程序信息。WebApp 的设计者应该预测出用户可能会请求下载该驱动程序, 并且直接提供下载的导航辅助。

传达。界面应该能够传达由用户启动的任何活动的状态。传达可以是直接的(例如一条

提问 在设计图形用户界面时是否有须遵循的一些基本原则?

[337]

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具

② 本书对 Tognozzi 的最初准则进行了修改与扩展。这些原则的进一步讨论参见 [Tog01]。

文本消息),也可以是间接的(例如在打印机中移动的纸张表明打印机正在工作)。

一致。导航控制、菜单、图标和美学风格(例如颜色、形状和布局)的使用应该在整个应用系统中保持一致。例如,如果一个移动 App 在显示屏的底部使用一组四个图标(用来表示主要功能),这些图标应该出现在每个屏幕上,并且不应移动到顶部来显示。图标的含义在应用范围内应该是不言而喻的。

自律。界面应该辅助用户在整个应用中移动,但也应该坚持使用已经为应用建立起来的导航习惯,以这样的方式来辅助用户。例如,对内容的导航应该受到用户 ID 和密码的访问控制,而不应该提供能使用户改变这种控制的导航机制。

效率。应用的设计和界面应该优化用户的工作效率,而不是优化设计与构建 WebApp 的 Web 工程师的效率,也不是优化运行系统的客户/服务器环境的效率。Tognozzi[Tog01]在讨论这一问题时写道:“这个简单的事实就是,为什么对于参与软件项目的每个人来说,认识到提高用户生产率的重要性及理解开发有效率的应用和提高用户效率的根本区别是非常重要的。”

引述 最好的旅程应该有最少的步骤,能够缩短用户和他们要到达的目标之间的距离。

作者不详

338

灵活性。界面应该足够灵活,既能够使其中一些用户直接完成任务,也能够使另一些用户以一种比较随意的方式浏览应用。在每种情况下,界面能够使用户认识到自己在哪里,并且给用户提提供撤销错误及从选错的导航路径返回的功能。

关注点。界面(及界面表示的内容)应该关注用户正在完成的任务。这个概念对移动 App 来说格外重要,如果设计师试图做得太多会使得界面变得非常杂乱。

人机界面对象。对于 WebApp 和移动 App,已经开发了大量可复用的人机界面对象库。使用这些对象库。能被最终用户“看到的、听到的、接触到的以及用别的方式感知到的”[Tog01]任何界面对象都能从大量对象库的任何一个中获得。

缩短等待时间。应用不应该让用户等待内部操作的完成(例如,下载一个复杂的图形图像),而应该利用多任务处理方式,从而使用户继续他的处理工作,看起来就像前面的操作已经完成一样。除了减少等待时间,如果有延迟事件发生,则必须通知用户,从而使用户知道正在发生的事情,包括:(1)在选中选项后,如果应用没有立即做出响应,则应该提供声音反馈;(2)显示一个动态时钟或进度条表示处理工作正在进行中;(3)当处理过程很长时,提供娱乐活动(例如动画或文本演示)。

学习能力。应用应将学习时间减到最少,并且一旦已经学习过了,当再次访问此应用时,将所需要的再学习时间减到最少。一般而言,界面应该侧重于简单、直观的设计,将内容和功能分类组织,这样对于用户来说很直观。

隐喻。只要隐喻适合应用和用户,使用交互隐喻的界面就更容易学习和使用。隐喻应该采用用户熟悉的图片和概念,但是并不要求是现实生活的精确再现。

建议 隐喻是一种出色的想法,因为隐喻能够反映现实世界的经验。只是要确保你选择的隐喻是最终用户所熟悉的。

易读性。界面展示的所有信息对于老人和年轻人都应该是易读的。界面设计者应该着重选择易读的字型式样、字体大小以及可以增强对比效果的背景颜色。

跟踪状态。在合适的时候,应该跟踪和保存用户状态,使得用户能够退出系统,稍后返回系统时又能回到退出的地方。一般而言,可设计 cookies 来存储状态信息。然而,cookies 是一种备受争议的技术,别的设计方案也许对某些

用户来说更合适。

可见的导航。设计合理的界面提供了这样的设想,“用户待在同一个地方,工作被带到他们面前”[Tog01]。使用这种方法后,导航就不再是用户关心的事情了,用户检索内容对象,并选择功能,这些功能都是通过界面显示并执行的。

339

SafeHome 界面设计评审

[场景] Doug Miller 的办公室。

[人物] Doug Miller, SafeHome 软件工程师团队经理; Vinod Raman, SafeHome 产品软件工程师团队成员。

[对话]

Doug: Vinod, 你和你的团队是否有可能评审 SafeHomeAssured.com 电子商务的界面原型?

Vinod: 是的……我们所有人都从技术角度对它进行了仔细检查,而且我还做了一些记录。

昨天我将这些记录发给了 Sharon (SafeHome 电子商务网站外包供应商 Web 工程团队的经理)。

Doug: 你和 Sharon 可以在一起详细讨论一下……给我一份重要问题的总结。

Vinod: 总的来说,他们已经做得很好了,没有遇到什么阻力。但是,这是一个典型的电子商务界面,具有高雅的美学设计、合理的布局设计。他们已经完成了所有重要功能……

Doug (可怜地微笑): 但是?

Vinod: 是的,有些小问题。

Doug: 例如……

Vinod (给 Doug 看界面原型的序列情节故事板): 这是一些显示在主页上的主要功能菜单。

学习 SafeHome

描述你的住宅

获得 SafeHome 构件建议

购买 SafeHome 系统

获得技术支持

问题并不在于这些功能,它们都没有问题,但是抽象级别不太合适。

Doug: 它们是主要功能,对吗?

Vinod: 没错。但是有这样一个问题,你可以通过输入构件列表来购买系统,如果你不想描述房子,就没有必要描述。我建议在主页上创建 4 个菜单选项:

学习 SafeHome

确定你所需要的 SafeHome 系统

购买 SafeHome 系统

获得技术支持

当你选择了“确定你所需要的 SafeHome 系统”时,你会有下面的选项:

选择 SafeHome 构件

获得 SafeHome 构件建议

如果你是一个有经验的用户,那么将从一组分好类的下拉菜单中选择构件,包括传感器、摄像机、控制面板等。如果需要帮助,可以请求系统提供建议,那时系统需要你描述一下你的房间。我认为这样更合理。

Doug: 我同意。关于这个问题你和 Sharon 谈过了吗?

Vinod: 没有,我想先和市场部讨论一下,然后我会给她打电话。

Nielsen 和 Wagner[Nie96] 提出了一些实际可行的界面设计指导原则(基于他们对重要 WebApp 的重新设计),这些原则很好地补充了本节前面提出的准则:

- 不要迫使用户阅读大量的文本信息,特别是当文本的内容是解释 WebApp 的操作,

或者是辅助导航时。

- 除了不可避免的操作外，不要让用户进行滚动操作。
- 设计不应该依赖于浏览器的功能来辅助导航。
- 美学效果绝不应该取代功能性。
- 不要强迫用户搜索显示如何链接到其他内容或服务。

好的界面设计能够提高用户对网站提供的内容或服务的理解程度，它并不一定要有闪烁的动画，但总应该是结构良好的，且具有符合人机工程学的声音。

引述 人们对于那些设计糟糕的 WWW 站点几乎没有耐心。

Jakob Nielsen,
Annette Wagner

15.5.2 WebApp 和移动 App 的界面设计 workflow^①

本章前面曾经提到用户界面设计首先要确定用户、任务和环境需求。一旦确定了用户任务，就可以创建和分析用户场景（用例），并定义一组界面对象和活动。

需求模型包含的信息构成了创建屏幕布局的基础，屏幕布局描述图标、图形设计和位置、描述性屏幕文本的定义、窗口标题定义及规格说明、主菜单和子菜单项目的规格说明。接着使用工具创建原型，并最终实现用户界面模型。下面的任务代表了一个基本工作流程：

1. 对需求模型中的信息进行评审，并根据需要进行优化。
2. 开发 WebApp 界面布局的草图。如果界面布局已经存在（在需求建模时开发了一个原型），则应该根据需要对其进行评审和优化。
3. 将用户目标映射到特定的界面行为。对于大多数 WebApp 和移动 App 来说，用户的主要目标相对比较少。应该将这些目标映射到特定的界面行为，如图 15-4 所示。实际上，界面设计人员必须回答下面的问题：“界面是如何让用户完成每个目标的？”

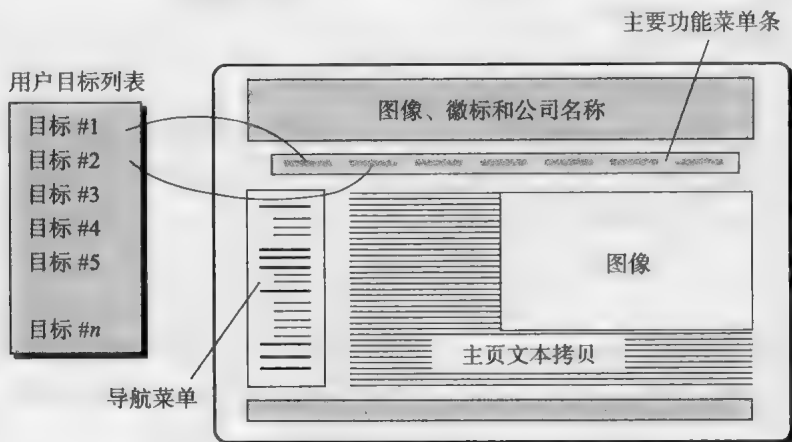


图 15-4 将用户目标映射到特定的界面行为

4. 定义与每个行为相关的一组用户任务。每个界面行为（例如购买商品）与一组用户任务相联系。在分析建模的过程中已经确定了这些任务。在设计期间，它们必须与明确的交互建立对应关系，这些交互包括导航事件、内容对象和应用功能。
5. 为每个界面行为设计情节故事板映像。在考虑每种行为时，应该创建序列情节故事板映像（屏像），来描述界面是怎样响应用户的交互行为的。应该明确内容对象（即使它们还没有设计和开发），并确定导航链接。

① 在第 17 章和 18 章将更详细地讨论 WebApp 和移动 App。

6. 利用美学设计中的输入来细化界面布局和情节故事板。在大多数情况下, 粗略的布局和情节故事板是由 Web 工程师完成的, 但是重要商业网站的美学外观通常是由专业绘图师而不是技术专家完成的。
 7. 明确实现界面功能的界面对象。这一任务可能需要在现有对象库中搜索, 找到那些适合界面的可复用对象(类)。另外, 在此时定义任何需要的自定义类。
 8. 开发用户与界面交互的过程表示。这一可选的任务利用 UML 顺序图或活动图(附录 1)描述用户与 WebApp 交互时的活动流(和决策流)。
 9. 开发界面的行为表示法。这一可选的任务利用 UML 的状态图(附录 1)表示状态转换和引起状态转换的事件, 并定义控制机制(即通过用户可用的对象和行为改变一个应用系统的状态)。
 10. 描述每种状态的界面布局。利用在任务 2 和任务 5 中开发的设计信息, 把确定的布局和屏幕图像与任务 8 中描述的每个 WebApp 状态联系起来。
 11. 优化和评审界面设计模型。界面的评审应该以可用性为重点。
- 值得注意的是, Web 工程团队所选择的最终任务集必须适合待构建应用的特殊需求。

15.6 设计评估

一旦建立好可操作的用户界面原型, 必须对其进行评估, 以确定满足用户的需求。评估可以从非正式的“测试驱动”(比如用户可以临时提供一些反馈)到正式的设计研究(比如向一定数量的最终用户发放评估问题表, 采用统计学的方法进行评估)。

[342]

用户界面评估的循环如图 15-5 所示。完成设计模型后就开始建立第一级原型; 用户对该原型进行评估^①, 直接向设计者提供有关界面功效的建议, 如采用正式的评估技术(比如使用提问单、分级评分表), 这样设计者就能从调查结果中得到需要的信息(比如 80% 的用户不喜欢其中保存数据文件的机制); 针对用户的意见对设计进行修改, 完成下一级原型。评估过程不断进行下去, 直到不需要再修改为止。

原型开发方法是有效的, 但是否可以在建立原型以前就对用户界面的质量进行评估呢^②? 如果能够及早地发现和改正潜在的问题, 就可以减少评估循环执行的次数, 从而缩短开发时间。界面设计模型完成以后, 就可以运用下面的一系列评估标准 [Mor81] 对设计进行早期评审:

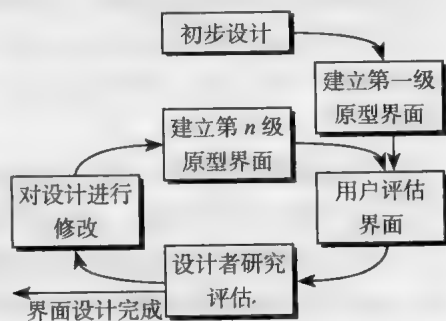


图 15-5 界面设计评估循环

1. 系统及其界面的需求模型或书面规格说明的长度和复杂性在一定程度上体现了用户学习系统的难度。
2. 指定用户任务的个数以及每个任务动作的平均数在一定程度上体现了系统的交互时间和系统的总体效率。
3. 设计模型中动作、任务和系统状态的数量体现了用户学习系统时所记忆内容的

① 注意, 人类工程学和界面设计方面的专家也可对界面进行审查。这些审查叫作启发评估或认知走查。

② 一些软件工程师更倾向于开发一个简单描画设计的用户界面模型, 称之为纸上原型, 使相关人员在交付任何程序资源之前先测试验证 UI 的内容。具体过程参见 http://www.paperprototyping.com/what_examples.html。

343

多少。

4. 界面风格、帮助设施和错误处理协议在一定程度上体现了界面的复杂度和用户的接受程度。

一旦第一个原型完成以后,设计者就可以收集到一些定性和定量的数据以帮助进行界面评估。为了收集定性的数据,可以进行问卷调查,使用户能够评估界面原型。如果需要得到定量数据,就必须进行某种形式的定期研究分析。观察用户与界面的交互,记录以下数据:在标准时间间隔内正确完成任务的数量、使用动作的频率、动作顺序、观看屏幕的时间、出错的数目、错误的类型、错误恢复时间、使用帮助的时间、标准时间段内查看帮助的次数。这些数据可以用于指导界面修改。

有关用户界面评估方法的详细论述已超出了本书的范围,有兴趣的读者可以参考 [Hac98] 和 [Sto05] 等文献。

15.7 小结

用户界面可以说是计算机系统或产品的最重要元素。糟糕的界面设计可能会严重地阻碍用户挖掘系统的计算能力,也会阻碍用户挖掘应用程序的信息内容。事实上,即使应用具有良好设计和可靠实现,糟糕的界面也可能导致该系统失败。

三个重要的原则可用于指导有效的用户界面设计:(1)把控制权交给用户;(2)减少用户的记忆负担;(3)保持界面一致性。为了得到符合这些原则要求的界面,必须实施有组织的设计过程。

用户界面的开发首先从一系列的分析任务开始。用户分析确定了各类最终用户的概况,并且使用了从各种业务和技术资源收集来的信息。任务分析定义了用户任务和行为,其中使用了细化或面向对象的方法、用户用例的应用、任务和对象的细化、工作流分析和层级任务表示等,来获得对人机交互的充分理解。环境分析可厘清界面必须操作的物理结构和社会结构。

一旦任务确定下来,就可以通过创建和分析用户场景来定义一组界面对象和动作。这为创建屏幕布局提供了基础,屏幕布局描述了图形的设计和图标的放置、描述性屏幕文字的定义、窗口的规格说明和标题以及主菜单和子菜单项规格说明。诸如响应时间、命令和动作结构、错误处理和帮助设施等设计问题应该在细化设计模型时考虑。很多实现工具可以用于创建供用户评估的原型。

344

像传统软件的界面设计一样,WebApp 和移动 App 界面的设计体现了用户界面的组织结构和屏幕的布局,是对交互模式的定义,也是对导航机制的描述。在设计布局和界面控制机制时,界面设计原则和界面设计工作流为 WebApp 或移动 App 设计者提供了向导。

用户界面是软件的窗口。在很多情况下,界面塑造了用户对系统质量的感知。如果这个“窗口”污点斑斑、凹凸不平或破损不堪,用户也许会选择其他更有效的计算机系统。

习题与思考题

- 15.1 描述一下你操作过的最好和最差的系统界面,采用本章介绍的相关概念对其进行评价。
- 15.2 在 15.1.1 节的基础上,再给出两条“把控制权交给用户”的设计原则。
- 15.3 在 15.1.2 节的基础上,再给出两条“减轻用户的记忆负担”的设计原则。
- 15.4 在 15.1.3 节的基础上,再给出两条“保持界面一致”的设计原则。

15.5 考虑下面几个交互应用(或者导师布置的应用):

- a. 桌面发布系统。
- b. 计算机辅助设计系统。
- c. 室内设计系统(如 15.3.2 节所描述的)。
- d. 大学课程自动注册系统。
- e. 图书管理系统。
- f. 基于网络的公共选举投票系统。
- g. 家庭银行系统。
- h. 导师布置的交互应用。

对上面给出的每个系统,开发用户模型、设计模型、心理模型和实现模型。

15.6 选择习题 15.5 中所列的任何一个系统,使用细化或面向对象的方法进行详细任务分析。

15.7 在 15.3.3 节提供的内容分析列表中至少再添加 5 个问题。

15.8 继续做习题 15.5,为你所选择的应用定义界面对象和动作。确定每个对象类型。

15.9 对于在习题 15.5 中所选的系统,开发一组带有主菜单和子菜单项定义的屏幕布局。

15.10 针对 SafeHome 系统,开发一组带有主菜单和子菜单项的屏幕布局,可以选择一种不同于图 15-3 的方法。

15.11 对于在习题 15.5、习题 15.7 和习题 15.8 中所完成的任务分析设计模型和分析任务,描述你采用的用户帮助设施。

15.12 举例说明为什么反应时间变动是一个问题。

345

15.13 开发一种能自动集成错误消息和用户帮助设施的方法。即系统能自动识别错误类型,并提供帮助窗口,给出改正错误的建议。进行合理且完整的软件设计,其中要考虑到合适的数据结构和算法。

15.14 开发一个界面评估提问单,其中包括 20 个适用于大多数界面的通用问题。由 10 名同学完成你们所有人使用的交互系统的提问单。汇总你们的结果,并在班上做介绍。

扩展阅读与信息资源

尽管 Donald Norman 的著作(《The Design of Everyday Things》, reissue edition, Basic Books, 2002)不是专门阐述人机界面的,但其中涉及了进行有效设计的心理学,可以应用于用户界面的设计。对于那些非常关心高质量用户界面设计的人员,我们推荐此读物。Weinschenk 的著作(《100 Things Every Designer Should Know About People》, New Riders, 2011)并不特别侧重于软件,而是富有洞察力地提出了以用户为中心的设计。Johnson 的著作(《Designing with the Mind in Mind》, Morgan Kaufman, 2010)利用认知心理学开发有效的界面设计规则。

图形用户界面在现代计算世界中是无处不在的,无论是在 ATM、移动电话、汽车电子仪表盘、Web 站点,或者是在商业应用中,用户界面都为软件提供了窗口。正因如此,关于界面设计的书籍有很多,其中值得借鉴的有:

Ballard, 《Designing the Mobile User Experience》, Wiley, 2007。

Butow, 《User Interface Design for Mere Mortals》, Addison-Wesley, 2007。

Cooper 和他的同事, 《About Face 3: The Essentials of Interaction Design》, 3rd ed., Wiley, 2007。

Galitz, 《The Essential Guide to User Interface Design》 3rd ed., Wiley, 2007。

Goodwin 和 Cooper, 《Designing for the Digital Age: How to Create Human-Centered Products and Services》, Wiley, 2009。

Hartson 和 Pyla, 《The UX Book: Process and Guidelines For Ensuring a Quality User Experience》, Morgan Kaufman, 2012。

Lehikonen 和他的同事, 《Personal Content Experience : Managing Digital Life in the Mobile Age》, Wiley-Interscience, 2007。

Nielsen, 《Coordinating User Interfaces for Consistency》, Morgan-Kaufmann, 2006。

Pratt 和 Nunes, 《Interactive Design》, Rockport, 2013。

Rogers 和他的同事, 《Interactive Design: Beyond Human-Computer Interaction》, 3rd ed., Wiley, 2011。

Shneiderman 和他的同事, 《Designing the User Interface : Strategies for Effective Human-Computer Interaction》, 5th ed., Addison-Wesley, 2009。

Tidwell, 《Designing Interfaces》, O'Reilly Media, 2nd ed., 2011。

Johnson 的著作 (《GUI Bloopers: Common User Interface Design Don'ts and Do's》, 2nd ed., Morgan Kaufmann, 2007 和 《GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers》, Morgan Kaufmann, 2000) 对那些通过检查反例来实现高效学习的人提供了有用的指导。Cooper 编写的广受欢迎的书 (《The Inmates Are Running the Asylum》, Sams Publishing, 2004) 讨论了为什么高技术产品常令人感到疯狂, 以及如何设计不让人疯狂的产品。

在网上有大量关于用户界面设计的信息, 有关用户界面设计的最新参考文献列表可在 SEPA Web 站点 www.mhhe.com/pressman 找到。

基于模式的设计

要点浏览

概念: 对于清晰描述的一组问题, 基于模式的设计通过查找一组已被证明有效的解决方案来创建新的应用。每个问题及其解决方案都用一个设计模式来描述, 事实上, 在此之前曾有其他软件工程师在设计其他应用时, 已经遇到了这个问题并实现了解决方案, 因此将这个设计模式编入目录并做了检验。每个设计模式都提供了一种已证明的方法, 用于解决某一部分问题。

人员: 软件工程师在遇到一个新的应用问题时, 通过搜索一个或多个模式存储库来寻找相关的解决方案。

重要性: 你是否听说过“重复发明轮子”这样的话? 这种情况在软件开发领域一直都存在, 既浪费时间也浪费精力。通过使用已经存在的设计模式, 可以获得特定问题的已被证明的解决方案。随着每种模式的应用, 解决方案被集成到完整的设计方案中, 所构建的应用就向着

完整的设计又迈进了一步。

步骤: 为了划分要解决的一系列问题的层次, 需要检查需求模型。将问题空间分割, 可以确定与软件功能和特性相关的问题子集。还可以对这些问题进行如下分类: 体系结构、构件级、算法、用户界面等。一旦定义了问题子集, 就查找一个或多个模式库, 以确定在适当的抽象层次上是否有设计模式存在。可以针对待构建软件的特定要求, 对选择使用的设计模式进行修改。如果找不到合适的模式, 则可以自定义问题的解决方案。

工作产品: 开发的设计模型用来描述体系结构、用户界面以及构件级的细节。

质量保证措施: 由于每个设计模式都要被转化成设计模型的一些要素, 因此要对工作产品进行评审, 以检查清晰性、正确性、完整性、与需求的一致性以及工作产品之间的一致性。

我们每个人都遇到过设计问题, 并且会想: 是否已经有人研究出这个问题的解决方案? 对这个问题的回答几乎总是肯定的! 问题是要找到解决方案, 确保此方案适合解决所遇到的问题, 理解限制此解决方案应用方式的约束, 最后将建议的解决方案应用到软件开发的设计环境中。

但是, 如果以某种方式规范解决方案会怎么样呢? 如果有某种描述问题的标准方式(使人们可以查找), 并用有组织的方法来描述问题的解决方案结果会如何呢? 结果是可以利用标准化的模板来规范和描述软件问题, 并提出问题(连同约束)的解决方案。所谓设计模式, 是用来描述问题以及解决方案的规范化方法, 在某种程度上允许软件工程界获取设计知识, 使解决方案能够得到重用。

关键概念

- 体系结构模式
- 行为型模式
- 构件级设计模式
- 创建型模式
- 设计错误
- 设计模式
- 框架
- 粒度
- 模式的类型
- 模式语言

软件模式的前身不是由计算机科学家首先提出来的，而是由建筑师 Christopher Alexander 提出来的，他发现在设计建筑物时总会遇到一系列重复的问题，于是他把这些重复出现的问题以及解决方案定义为模式，可以用下面的方式来描述模式 [Ale77]：

每个模式都描述了在我们所处环境内反复出现的问题，然后描述该问题的核心解决方案，这样你就可以几百万次地重复使用该解决方案，而不必用同样的方式重复工作两次。

Gamma[Gam95]、Buschmann[Bus96] 以及他们的许多同事首次在著作中将 Alexander 的思想引入到软件领域^①，现在已经有很多模式存储库，基于模式的设计可以用于很多不同的应用领域。

16.1 设计模式

设计模式可以描述为“表示特定上下文、问题和解决方案三者之间关系的三部分规则” [Ale79]。对于软件设计，上下文使读者理解问题所发生的环境，以及在此环境中什么样的解决方案才适合。一组需求（包括限制和约束）起到影响因素的作用，会影响如何在此问题的上下文环境中对问题进行解释以及如何有效地应用解决方案。

有理由认为大多数问题都有多种解决方案，但是只有适合问题所处环境的解决方案才是有效的。影响因素促使设计者去选择某一特定的解决方案。目的是提供一个最能满足影响因素的解决方案，即使这些影响因素是矛盾的。最后要说明的是，每种解决方案都有各自的结果，这些结果会影响软件的其他方面，并且在较大的系统中，对于要解决的其他问题，这些结果本身又会成为影响因素的一部分。

Coplien [Cop05] 用下面的方式描述了有效的设计模式的特点：

- 设计模式可以解决问题。模式可以捕捉解决方案，不只是抽象的原则或策略。
- 设计模式是已经得到验证的概念。模式借助于踪迹记录捕捉解决方案，而不是根据理论或猜测。
- 解决方案并不明显。很多问题解决技术（例如软件设计范型或方法）都试图从基本原理导出解决方案。最好的模式会间接产生问题的解决方案——对于大多数困难的设计问题而言，这是必要的方法。
- 设计模式描述关系。模式不只是描述模块，而且描述更深层的系统结构和机制。
- 模式具有显著的人性化元素（将人工干预降到最少）。所有软件都是为改善人类的生活舒适程度或生活质量而服务的，最好的模式明确要求具有美学和实用性。

设计模式可以使你免于“重复发明轮子”，更糟的是，发明的“新轮子”不是很圆，对于其使用目的来说太小，对于行驶的路面来说太窄。如果能有效地使用设计模式，你一定会成为更好的软件设计师。

关键概念

模式组织表
结构型模式
影响因素系统
用户界面设计
模式
WebApp 设计
模式

关键点 影响因素是限制以某种方式进行设计的那些问题的特征及解决方案的属性。

引述 我们的责任是尽力去做、去学习，改进解决方案，并传递下去。

Richard P.
Feynman

① 早期关于软件模式的讨论确实存在，但是这两本经典的书首次对这个主题进行了集中论述。

16.1.1 模式的种类

软件工程师对设计模式感兴趣甚至着迷的原因之一是由于人类天生善于模式识别。如果不是这样，我们就会停滞在空间和时间中——不能吸取过去的经验，不愿冒险前进，因为我们无法识别那些可能引起高风险的情况，而且让我们发疯的是，世界似乎没有规律或逻辑一致性。幸运的是，不会有这些情况发生，因为在现实生活的各个方面，实际上，我们都能识别模式。

在现实中，我们识别的模式来自人生经验。我们可以立即识别并理解模式的含义，而且清楚如何使用它。有些模式可以使我们深入了解重复出现的现象。例如，你正在下班回家的州际公路上，导航系统或是收音机通知你，在相反方向的路上发生了一起严重事故。你距离事故发生地点4公里，但是已经看到交通放缓，这种模式称为 RubberNecking。旅游车道的人慢慢向你的方向移动过来，以便看清楚高速路的对面究竟发生了什么事情。RubberNecking 模式很明显会产生可以预料的结果（交通堵塞），但它只是描述现象而已。用模式的术语来说，称为无生产力（nongenerative）的模式，因为它只是描述问题和背景，但不提供明确的解决方案。

考虑软件设计模式时，我们总是力图识别和记载有生产力（generative）的模式。也就是说，我们识别能描述系统中重要的和可重复方面的模式，这样的模式在影响因素（对于给定的上下文环境是独一无二的）内为我们提供了一种方式来构造重要的和可重复的方面。在理想的环境下，一组有生产力的设计模式能够用于“产生”一个应用系统或一个基于计算机的系统，其体系结构能够使系统适应变更。有时我们称之为生产性，“就是连续应用几种模式，每种模式都封装自己的问题和影响因素，更大的解决方案是以一些较小的解决方案的形式间接地表现出来的”[App00]。

设计模式所涉及的抽象和应用的范围很广。体系结构模式描述了很多可以用结构化方法解决的设计问题。数据模式描述了重现的面向数据的问题以及用来解决这些问题的数据建模解决方案。构件模式（也认为是设计模式）涉及与开发子系统和构件相关的问题、它们之间相互通信的方式以及它们在一个较大的体系结构中的位置。界面设计模式描述公共用户界面问题及具有影响因素（包括最终用户的具体特征）的解决方案。

WebApp 模式解决构建 WebApp 时遇到的问题，而且往往包括很多前面提到的一些其他模式。移动模式通常描述在开发移动平台的解决方案时遇到的问题。在较低的抽象层，习惯用语（idioms）描述如何在特定的编程语言环境下实现软件构件的全部或部分特定算法或数据结构。

Gamma 和他的同事^①[Gam95] 在关于设计模式的具有重大影响的书^②中，着眼于与面向对象设计相关的三种模式：创建型模式、结构型模式及行为型模式。

创建型模式着眼于对象的“创建、组合及表示”，并且提供了一种机制，使对象实例在一个系统内更容易生成，并坚持“在一个系统内创建的对象类型及数量方面的约束”[Maa07]。结构型模式着眼于如何将类和对象组织和集成起来，以创建更大结构的问题和解决方案。行为型模式解决

关键点 有生产力的模式描述了问题、环境和影响因素，而且描述了问题的实际解决方案。

349

建议 不要强行使用一个模式，即使它解决了眼前的问题。如果环境和影响因素是错误的，那么就寻找另一个模式。

提问 你有什么办法划分模式的类型吗？

^① 在模式文献中，Gamma 和他的同事 [Gam95] 常被称为“四人帮”（GoF）。

与对象间任务分配以及影响对象间通信方式有关的问题。

信息栏 创建型模式、结构型模式和行为型模式

大量适合于创建型、结构型和行为型类别的设计模式已经出现，并可以在网上找到。Wikipedia (<http://en.wikipedia.org/wiki/Software-design-pattern>) 记录了下面的模式。

创建型模式

- 抽象工厂模式：集中决定实例化什么工厂。
- 工厂方法模式：集中创建某一特定类型的对象，并从几种实现中选择其中的一种。
- 生成器模式：将一个复杂对象的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。

结构型模式

- 适配器模式：将一个类的接口转换成客户希望的另外一个接口。
- 聚集模式：是组合模式的一个版本，采用将产物聚集在一起的方法。
- 复合模式：复合模式就是一个具有同样接口的处理对象的树结构模式。
- 容器模式：创建对象的唯一目的就是装

载其他对象并管理它们。

- 代理模式：一个类的作用是作为另一个类的接口。
- 管道和过滤器：是一个过程链，每个过程的输出是下一个过程的输入。

行为型模式

- 责任链模式：对命令对象进行处理，或者通过逻辑包含的处理对象传递给其他对象进行处理。
- 命令模式：命令对象把行动和参数封装起来。
- 迭代器模式：迭代器用于按顺序访问一个聚集对象中的元素，而不必暴露其底层表示。
- 中介者模式：对于子系统中的一组接口，提供一个统一的接口。
- 访问者模式：将算法从一个对象中分离出来的方法。
- 层次访问者模式：提供一种方式，访问层次数据结构（例如树）上的每个节点。可以通过链接 www.wikipedia.org 获得每个模式的全面描述。

350

16.1.2 框架

模式本身可能不足以开发一个完整的设计。在某些情况下，可能需要为设计工作提供与实现相关的架构基础设施，称为框架。也就是说，可以选择“一个可重用的微型体系结构，此体系结构在某种上下文环境中为一系列软件抽象提供了通用的结构和行为……并在给定的领域内规定了这些结构和行为之间的协作和使用”[Amb98]。

框架不是体系结构模式，而是一个具有“插入点”（也称为钩子和插槽）集合的架构，可以适应特定的问题域。插入点使得软件工程师能够在架构内集成特定问题的类或功能。在面向对象的环境下，框架是相互协作的类的集合。

Gamma 和他的同事 [Gam95] 这样描述设计模式和框架之间的区别：

1. 设计模式比框架更抽象。框架可以用代码表示出来，但只有模式的实例才可以用代码表示出来。框架的长处是可以用编程语言编写，不仅可以用来研究，也可以直接执行

关键点 框架是可以重复使用的“微型体系结构”，可以作为应用其他设计模式的基础。

和重用……

2. 设计模式是比框架更小的体系结构元素。一个典型的框架包括一些设计模式，而设计模式却不包括框架。
3. 对设计模式的研究要比框架少。框架总是有特定的应用领域。与此相反，设计模式几乎可以用在任何类型的应用问题中。当然很可能有更多专用的设计模式，即使这些设计模式不能确定应用的体系结构。

[351]

实际上，框架设计师认为，在限定的应用领域内，可重用的微型体系结构适用于所有软件的开发。为了实现最大效率，框架可以不进行任何修改而直接使用。可能还需要增加更多的设计元素，但设计师只能通过插入点来充实框架。

16.1.3 描述模式

基于模式的设计开始于识别要构建的应用中的模式，然后进行查找，确定别人是否已经论述过这个模式，最后采用适合当前问题的模式。其中第二个任务常常是最困难的。那么怎样才能找到所需要的模式呢？

对这个问题的回答取决于以下四个方面的有效交流，即模式要解决的问题、所在的环境、环境的影响因素及所提出的解决方案。为了清楚无误地交流这些信息，需要描述模式的标准格式或模板。虽然已经提出了很多不同的模式模板，但几乎所有的模式模板都包含 Gamma 和他的同事 [Gam95] 所建议的主要内容。简化的模式模板如下所示。

引述 模式是不成熟的——意味着需要自己去完成，并使之适应你的环境。

Martin Fowler

信息栏 设计模式模板

模式名称——以简短但富于表现力的名字描述模式的本质。

问题——描述模式涉及的问题。

动机——提供问题的实例。

环境——描述问题所在的环境，包括应用领域。

影响因素——列出影响问题解决方式的全部影响因素，包括必须要考虑的限制和约束的讨论。

解决方案——提供问题解决方案的详细描述。

目的——描述模式以及模式所做的工作。

工作。

协作——描述其他模式对解决方案的贡献。

效果——描述实现模式时必须考虑的可能要做的折中以及使用模式的效果。

实现——在实现模式时，确定应该考虑的特殊问题。

已知应用——提供了设计模式在实际应用中的使用实例。

相关模式——相关设计模式的交叉索引。

应该仔细选择设计模式的名称。基于模式的设计的关键技术问题之一是在成千上万在候选模式中无法找到现有的模式。有意义的模式名称非常有助于搜索“正确”的模式。

模式模板为描述设计模式提供了标准化的方法。每个模板项都表现了要查找的设计模式的特征（例如通过数据库查找），从而能够找到合适的模式。

[352]

16.1.4 模式语言和模式库

提到语言这个词时,我们首先会想到的是自然语言(例如,英语、西班牙语、汉语)或程序设计语言(例如,C++、Java)。这两种语言都有语法和语义,用于以有效的方式交流思想或过程指令。

语言这个术语用于设计模式环境时意义稍有不同。模式语言包括模式集合,每个模式都用标准化的模板(16.1.3节)来描述并相互关联,以显示这些模式如何通过相互合作来解决应用领域中的问题。^①

在自然语言中,将单词组织成句子来表达意思。句子的结构通过语言的语法来描述。在模式语言中,以某种方式来组织设计模式,这种方式提供了“在特定的领域内描述良好设计实践的结构化方法”。^②

在某种程度上,模式语言类似解决特定应用领域内问题的超文本指令手册。首先按层次来描述所考虑的问题域,从领域相关的宏观设计问题开始,然后将宏观问题细化成较低的抽象层次。实质上,在软件环境中,宏观设计问题倾向于体系结构问题,它涉及应用的总体结构以及服务于系统的数据或内容。将体系结构问题细化为更低的抽象层次,从而导出解决子问题的设计模式,以及在构件(或类)级的相互协作。模式语言不是模式的顺序列表,而是一个内部相互连接的集合,在这个集合中,用户从宏观设计问题开始,“向下挖掘”以发现特定问题及其解决方案。

对于软件设计,人们已经提出了很多模式语言[Hil13]。软件设计包含设计模式,而设计模式是模式语言的一部分,通常存储在网络访问的模式库中。模式库提供了所有设计模式的索引,以及帮助用户理解模式间协作的超链接。

建议 如果找不到模式语言解决你的问题域,那么可以在另外的模式集中查找类似的模式语言。

网络资源 有用的模式语言列表参见 c2.com/ppr/titles.html。补充信息可以从 hillside.net/patterns/ 获得。

353

16.2 基于模式的软件设计

任何领域的顶级设计人员都具有一种神奇的能力,他们能看出表明问题特征的模式,并将相应的多个模式组合起来形成解决方案。在整个设计过程中,(当模式符合设计要求时)应该利用一切机会去寻找现成的设计模式,而不是去创建新模式。

16.2.1 不同环境下基于模式的设计

基于模式的设计不是脱离现实的。前面所讨论的体系结构设计、构件级设计及用户界面设计(第13~15章)的概念和技术都是与基于模式的方法联合使用的。

第12章所提到的一系列质量指导方针和属性可以作为所有软件设计决策的基础。决策本身是受许多基本设计概念(例如,关注点分离、逐步求精、功能独立)和最佳实践(例如,技术、建模表示方法)影响的。这些基本设计概念是利用经过了几十年演变的启发法获得的;作为构造的基础,最佳实践使设计更容易实施,也更为有效。

图16-1描述了基于模式的设计的步骤。软件设计师从描述系统抽象表示的需求模型(明确的或隐含的)开始工作。需求模型描述问题集合、建立上下文环境并明确主要影响因

① 模式语言最初是 Christopher Alexander 为建筑结构和城市规划提出来的。现在,模式语言已经发展到从社会科学到软件工程等各个方面。

② Wikipedia 关于模式语言的描述参见 http://en.wikipedia.org/wiki/Pattern_language。

素。需求模型只是用抽象的方式暗示了设计，但并不能明确表示设计。

当软件设计师开始工作时，牢记质量属性（第 12 章）总是很重要的。这些属性建立了评估软件质量的方法，但对实际获得软件质量的帮助很小。因此，要应用成熟的技术把需求模型中的抽象表示转化成更具体的形式，这就是软件设计。为了完成这项任务，应利用可以得到的体系结构设计、构件级设计及接口设计方法和建模工具，但只有当问题、环境及系统影响因素没有既有解决方案时才这样做。如果解决方案已经存在，就直接使用！这就意味着应用了基于模式的设计方法。

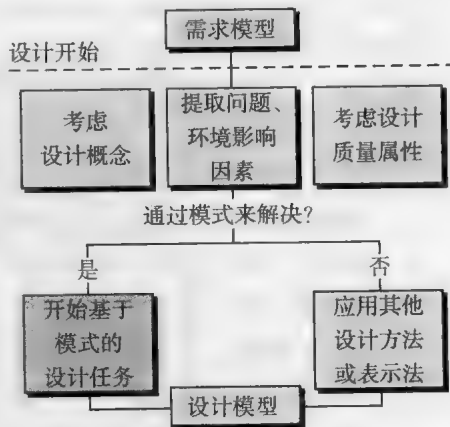


图 16-1 不同环境下基于模式的设计

16.2.2 用模式思考

基于模式的设计意味着“一种新的思考方式” [Sha05]，这种方式是从考虑环境（全局）开始的。在评价环境时，需要提炼出要解决问题的层次结构。其中的一些问题是全局性质的，其他问题要解决的是软件的特定性质和功能。影响因素会影响所有这些问题以及要提出的解决方案的性质。

为了使设计者用模式思考，Shalloway 和 Trott[Sha05] 提出了下面的方法：[⊖]

1. 保证理解全局——将要构建的软件所处的环境。需求模型表达了这一点。
2. 检查全局，提取在此抽象层上表示的模式。
3. 从“全局”模式开始设计，为将来的设计工作建立环境或架构。
4. “在环境的内部工作” [Sha05] 是在更低的抽象层上寻找有助于设计方案的模式。
5. 重复步骤 1～步骤 4，直到完成完整的设计。
6. 通过使每个模式适应将要构建的软件细节对设计进行优化。

提问 基于模式的设计关心的是要解决的问题。我怎样开始工作？

理解各个模式不是独立的实体是很重要的。处于高抽象层的设计模式常常影响较低抽象层的其他模式的应用方式。另外，模式间也经常相互协作。这意味着，当选择一个体系结构模式时，会对所选择的构件级设计模式有很大影响。同样，当选择一个特定的接口设计模式时，有时不得不使用与之协作的其他模式。

举例说明，考虑一下 WebApp SafeHomeAssured.com。如果想要考虑全局，这个 WebApp 必须考虑如下问题：如何提供关于 SafeHome 产品和服务的信息？如何将 SafeHome 产品和服务卖给顾客？如何建立基于网络的监控，并对已安装的安全系统进行控制？以上每个基本问题都可以进一步细化为一系列更小的子问题。

例如，如何通过因特网进行销售？这个问题就隐含着电子商务（E-commerce）模式，电子商务模式本身又包含着许多较低抽象层上的模式。电子商务模式（也可能是一个体系结构模式）意味着建立客户账户、展示要销售的产品、选择购买的产品等。因此，如果用模式思考，重要的是，要确定是否存在建立账户的模式。在问题所处的环境下，如果建立账户

⊖ 此方法基于 Christopher Alexander[Ale79] 的工作。

(SetUpAccount)是一个可以使用的可行模式,那么这个模式可能要与其他模式进行协作,例如 BuildInputForm、ManageFormsInput 及 ValidateFormsEntry。每个模式都描述了解决的问题以及可能应用的解决方案。

16.2.3 设计任务

使用基于模式的设计思想时,实施的设计任务如下:

1. **检查需求模型,并开发问题的层次结构。**通过分离问题、环境及相关的影响因素,描述每个问题及子问题。然后,从宏观问题(在较高的抽象层)到更小的子问题(在较低的抽象层)依次进行处理。
2. **针对问题域,确定是否已经开发了可靠的模式语言。**如 16.1.4 节所述,模式语言涉及与特定应用领域相关的问题。SafeHome 软件组应该去寻找专门为家庭安全产品开发的模式语言。如果不能找到这一级别的模式语言,软件组就应该将 SafeHome 软件问题划分成一系列通用问题域(例如,数字设备监控问题,用户界面问题,数字视频管理问题),并查找合适的模式语言。
3. **从宏观问题开始设计,确定是否存在适用的一个或多个体系结构模式。**如果一个体系结构模式是可用的,那么一定要检查所有相互协作的模式。如果这个模式合适,就对所提出的设计方案进行修改,并构建一个充分代表此模式的设计模型元素。例如, SafeHomeAssured.com WebApp 的宏观问题是用电子商务(E-commerce)模式(16.2.2 节)解决的。这个模式为解决电子商务的需求提出了一个特定的体系结构。
4. **利用为体系结构模式所提供的协作,检查子系统或构件级问题,并查找合适的模式解决这些问题。**有必要在其他模式库及与体系结构解决方案相对应的模式列表中进行查找。如果找到了合适的模式,就对所提出的设计方案进行修改,并构建一个充分代表此模式的设计模型元素。一定要应用步骤 7。
5. **重复步骤 2 ~ 步骤 4,直到所有的宏观问题都得到了解决。**即从全局开始,并逐渐在更详细的层次上解决问题。
6. **如果已经分离出了用户界面设计问题(几乎总是这种情况),那么为了找到合适的模式,查找多个用户界面设计模式库。**用类似于步骤 3、步骤 4 和步骤 5 的方式继续进行。
7. **不考虑抽象层的情况下,如果模式语言、模式库或单个的模式是存在的,那么将要解决的问题和现存的模式进行比较。**检查环境和影响因素,以确保模式提供的解决方案与问题相吻合。
8. **当从模式推导出设计后,使用设计质量标准作为参考,对设计进行优化。**

虽然上面列举的设计方法是自顶向下的,但是现实的设计方案有时要复杂得多。关于这一点, Gillis[Gil06]是这样论述的:

软件工程中的设计模式是以演绎和推理的方式使用的。设一个通用问题或需求是 X,模式 Y 可以解决 X,因此就使用模式 Y。然而从整个过程来考虑,我相信自己并不孤单——我发现它更有组织、更多是归纳而不是演绎、更多是自底向上而不是自顶向下。

另外,基于模式的设计方法必须与其他软件设计概念和技术配合使用。

16.2.4 建立模式组织表

随着基于模式的设计的发展,对来自许多模式语言和模式库中的候选模式进行组织和分

类是很麻烦的。为了方便对候选模式的评估进行组织，Microsoft[Mic13] 建议创建一张模式组织表，其一般格式如图 16-2 所示。

	数据库	应用	实现	基础设施
数据 / 内容				
问题陈述……	模式名称		模式名称	
问题陈述……		模式名称		模式名称
问题陈述……	模式名称			模式名称
体系结构				
问题陈述……		模式名称		
问题陈述……		模式名称		模式名称
问题陈述……				
构件级				
问题陈述……		模式名称	模式名称	
问题陈述……				模式名称
问题陈述……		模式名称	模式名称	
用户界面				
问题陈述……		模式名称	模式名称	
问题陈述……		模式名称	模式名称	
问题陈述……		模式名称	模式名称	

图 16-2 模式组织表 [Mic04]

通过采用图 16-2 所示的形式，模式组织表可以作为电子表格模型实现。问题陈述的简单列表分为数据 / 内容、体系结构、构件级以及用户界面问题，显示在左（阴影）列。上面一行列出了数据库、应用、实现和基础设施这四种模式类型。表格里注明了候选模式的名称。

为了提供组织表的条目，需要在模式语言和模式库中查找解决特定问题陈述的模式。当找到一个或多个候选模式时，找到问题陈述所对应的行，以及模式类型对应的列，填入模式名称。模式的名称作为超链接填入，链接到包含此模式完整描述的 Web 地址 URL。

358

16.2.5 常见设计错误

在使用基于模式的设计时，会发生很多常见的错误。在某些情况下，由于没有足够的时间去理解根本问题、问题所在环境及影响因素，所以，可能会选择看似正确但并不适合解决方案的模式。一旦选择了错误的模式，就会拒绝正视自己的错误，并强行适应模式。在另外的情况下，你所选择的模式没有考虑到问题的某些影响因素，导致了所选择的模式不理想或者是错误的。有时，只照着字面的意思去应用模式，而没有针对具体问题对模式进行修改。

建议 即使模式能解决手头的问题，也不要急于使用。如果环境和影响因素是错的，就要去寻找另一个模式。

这些错误能够避免吗？在大多数情况下，答案是“肯定的”。好的设计师会寻找其他意见，并欢迎对其工作进行评审。评审技术在第 20 章讨论，对于要解决的软件问题，评审技术有助于确保应用基于模式的设计能够得到高质量的解决方案。

16.3 体系结构模式

如果房屋建造者决定建造一个具有旧式建筑风格的中央门厅式建筑，那么可以应用的建筑风格就是唯一的。建筑风格的细节（例如，壁炉的数目，房子的外观以及门窗的位置等）可以有变化，但是一旦房屋的整体结构确定下来了，那么设计的风格就随之确定了^①。

体系结构模式与房屋建筑结构有点区别。例如，每户都要使用厨房（Kitchen）模式。厨房模式以及与其它合作的其他模式会涉及的问题有：食物的存储与准备，完成这些任务所需要的工具，以及这些与工作流程相关的工具在房间中的放置规则等。另外，该模式还会涉及的问题有：厨房台面、照明、墙壁开关、中心岛、室内地面等问题。显然，对于一个厨房，会存在多种设计，通常取决于环境和影响因素。但是，可以在厨房模式所建议的“解决方案”的环境内构思每个设计。

选择一个在特定领域具有代表性的体系结构模式之前，必须评估它在以下方面的适合程度：应用系统、总体的体系结构风格、模式指定的环境及影响因素。

关键点 软件体系结构中，可能有很多体系结构模式涉及并发性、持久性及分布性等问题。

359

信息栏 设计模式库

网上有很多设计模式的原始资料。一些模式可以从独立发布的模式语言中得到，另一些模式可以从模式门户或模式库中得到。下面的网络原始资料值得一看。

- Hillside.net (<http://hillside.net/patterns/>)。提供最全面的模式和模式语言集的网站之一。
- Portland Pattern Repository (Portland 模式库, <http://c2.com/ppr/index.html>)。包含各种模式资源和模式集的链接。
- Pattern Index (模式索引, <http://c2.com/cgi/wiki?PatternIndex>)。一个“精选的模式集”。
- Handbook of Software Architecture (软件体系结构手册, http://researcher.watson.ibm.com/researcher/view_project.php?id=3206/)。数百个体系结构和构件设计模式的参考书目。

用户界面模式集

- UI/HCI Patterns (UI/HCI 模式, <http://www.hcipatterns.org/patterns/borchers/>

patternIndex.html)。

- Jennifer Tidwell's UI Patterns (Jennifer Tidwell 的用户界面模式, <http://designinginterfaces.com/patterns/>)。
- Mobile UI Design Patterns (移动用户界面设计模式, <http://profs.info.uaic.ro/~evalica/patterns/>)。
- Pattern Language for UI Design (用户界面设计的模式语言, www.maplefish.com/todo/papers/Experiences.html)。
- Interaction Design Library for Games (游戏的交互设计库, <http://www.eelke.com/files/pubs/usabilitypatternsingames.pdf>)。
- UI Design Patterns (用户界面设计模式, www.cs.helsinki.fi/u/salaakso/patterns/)。

专业的设计模式

- Aircraft Avionics (机载航空电子设备, <http://g.oswego.edu/dl/acs/acs/acs.html>)。
- Business Information Systems (商业信息系统, www.objectarchitects.de/arcus/cookbook/)。

① 这意味着将有中央大厅和走廊，房间在大厅的左边和右边，房屋有两（或更多）层，卧室在楼上，等等。一旦决定采用旧式建筑风格的中央大厅，这些“规则”就确定了。

- Distributed Processing (分布式处理, www.cs.wustl.edu/~schmidt/)。
- IBM Patterns for e-Business (电子商务的 IBM 模式, www-128.ibm.com/developer-works/patterns/)。
- Yahoo! Design Pattern Library (雅虎设计模式库, <http://developer.yahoo.com/ypatterns/>)。
- WebPatterns.org (<http://www.welie.com/index.php>)。

16.4 构件级设计模式

构件级设计模式可以提供通过验证的解决方案, 这些解决方案可以解决从需求模型中提取的一个或多个子问题。在很多情况下, 这种类型的设计模式关注系统的某些功能元素。例如, SafeHomeAssured.com 应用必须解决下面的设计子问题: 我如何得到 SafeHome 设备的产品规格说明及相关信息?

在阐明了必须要解决的子问题后, 现在应该考虑影响解决方案的环境和影响因素。通过研究合适的需求模型用例, 会发现消费者通过使用 SafeHome 设备(例如安全传感器或摄像机)的规格说明来获取信息, 但是, 只有选择了电子商务功能后才能使用与规格说明有关的其他信息(例如定价)。

子问题的解决方案包括搜索。既然搜索是一个很常见的问题, 所以有很多与搜索有关的模式就并不奇怪。通过检查大量模式库, 会找到如下的模式, 以及每个模式解决的问题:

360

AdvanceSearch: 用户必须在大量选项里找到特定项。

HelpWizard: 用户在查找与网站有关的某一主题或者当用户想要在网站找到某一特定网页时需要帮助。

SearchArea: 用户必须找到一个网页。

SearchTips: 用户需要知道如何控制搜索引擎。

SearchResults: 用户必须处理搜索结果列表。

SearchBox: 用户必须找到一项或特定的信息。

对于 SafeHomeAssured.com 而言, 产品的数目并不是特别大, 且每个产品有一个相对简单的分类, 所以 AdvanceSearch 和 HelpWizard 也许不必要。同样, 搜索很简单, 不需要 SearchTips。下面给出了关于 SearchBox 的部分描述。

SearchBox

(根据 www.welie.com/patterns/showPattern.php?patternID=search 整理。)

问题: 用户需要找到一项信息或特定信息。

动机: 在任何情况下, 关键词搜索都作用于组织成网页的内容对象集合。

环境: 用户不会用导航去获取信息或内容, 而是在包含很多网页的内容里直接搜索。所有网站都有主导航系统。用户可能想在一个类里搜索一项, 也可能想进一步指定一个查询。

影响因素: 网站有主导航系统。用户可能想在一个类里搜索一项, 也可能通过简单的布尔运算符进一步指定一个查询。

解决方案: 搜索功能是由搜索标签、关键词字段、筛选程序(如果适用)以及“搜索”按钮构成的。按回车键和点击搜索按钮的功能是一样的。同时在一个单独的网页上提供了搜索提示和示例。进入这个网页的链接就放在搜索功能的旁边。编辑搜索词的编辑框足够容纳

3 个典型的用户查询（通常约 20 个字符）。如果筛选条件多于两个，可以用组合框，否则用单选按钮。

搜索结果显示在新的网页上，这个网页上有清除标签，而且至少含有“搜索结果”或类似的信息。位于网页顶部的带有输入关键词的搜索功能可以重复执行，使用户知道关键词是什么。

361

在 16.1.3 节中继续描述模式的其他条目。

接下来，模式要描述如何访问、表示、匹配搜索结果等。在此基础上，SafeHome-Assured.com 团队需要设计实现搜索的构件，或（更可能）去获取现有的可重用构件。

SafeHome 应用模式

[场景] SafeHomeAssured.com 通过因特网实现传感器控制的软件增量设计的非正式讨论。

[人物] Jamie（负责设计）和 Vinod（SafeHomeAssured.com 首席系统架构师）。

[对话]

Vinod：摄像机控制界面的设计进展得怎样？

Jamie：还好，没有太多问题。我已经完成了大部分连接实际传感器的功能，开始考虑用户界面，从远程网页实际推拉摇移摄像机，但是我不能肯定我没弄错。

Vinod：你想到了什么？

Jamie：嗯，要求是这样的，摄像机控制器需要高交互性——当用户移动控制器时，摄像机也要尽快移动。所以，我在想用一组按钮排列成普通的摄像机的样子，但当用户点击按钮时，就控制了摄像机。

Vinod：嗯。是的，它会工作，但我不确定会正确地工作——每次点击一个控制按钮时，你需要等待整个客户-服务器通信程序实现，所以无法得到快速反馈的良好体验。

Jamie：我也是这么想的，这也是我不喜欢这种方法的原因，但我不确定是否还有

别的做法。

Vinod：为什么不用 InteractiveDeviceControl 模式！

Jamie：那是什么？我没听说过。

Vinod：你所描述的问题基本上就是这个模式。它提出的解决方案基本上是建立服务器与设备的控制连接，通过发送控制命令完成，而不用发送通常的 HTTP 请求。它还可以指导你如何用一些简单的 AJAX 技术实现连接。一些简单的客户端 Java 脚本就可以直接和服务器通信并发送命令，让用户尽快完成任务。

Jamie：太棒了！那正是我需要用来解决这件事的方法。在哪能找到它？

Vinod：可以在联机库中得到。这里有网址。

Jamie：我会去查查看。

Vinod：是的，但是要记得去检查模式的结果字段。我似乎记得有关安全问题的某些事情需要特别小心。因为你建立了一个单独的控制通道，所以绕过了通常的网络安全机制。

Jamie：有道理！我可能还没有想到那一点！谢谢。

16.5 用户界面设计模式

近年来，人们已经提出了数百个用户界面 (UI) 模式，大部分模式属于 Tidwell[Tid02] 和

vanWelie[Wel01] 所描述的 10 类模式之一。几个有代表性的类别（用一个简单的例子讨论^①）如下：

Whole UI（整个用户界面）。为最高级结构及整个用户界面的导航提供设计指导。

- 模式：Top-Level Navigaion。
- 概述：用于网站或应用程序完成一些主要功能。提供了一个高级菜单，徽标或图形标识常常与菜单一起出现，通过菜单可以直接导航到系统的主要功能。
- 详细资料：主要功能（一般仅限于 4～7 个功能名）用一条水平的文字横列在显示器的上部（也可能是垂直纵列格式）。每个功能名提供到达相应的功能或信息源的链接。通常使用后面讨论的 Bread Crumbs 模式。
- 导航元素：每个功能或内容的名称代表连接相应功能或内容的链接。

Page layout（页面布局）。处理一般的页面组织（网站）或是不同的屏幕显示（交互式应用程序）。

- 模式：Card Stack。
- 概述：当必须按任意顺序选择与特征或功能相关的一些特定子功能或内容分类时，使用该模式。该模式提供了大量的选项卡，用鼠标点击可以选择，每个选项卡代表特定的子功能或内容分类。
- 详细资料：选项卡是容易理解的隐喻，且易于用户操作。每个选项卡的格式可能有些不同。一些选项卡可能需要输入信息，会有按钮或其他导航机制；另一些选项卡可能是提供信息的；该模式可结合 Drop-Down List、Fill-in-the-Blanks 等其他模式一起使用。
- 导航元素：用鼠标点击标签，就会显示相应的选项卡。选项卡的导航功能也可能同时出现，但是通常情况下，这些只是初始化与选项卡数据相关的功能，并不连接到其他显示。

Forms and input（表单和输入）。考虑完成表单级输入的各种设计技术。

- 模式：Fill-in-the-Blanks。
- 概述：将字母数字数据输入到“文本框”中。
- 详细资料：可以将数据输入到文本框中。一般地，通过选取某些文本或图形指示器（例如“搜索”“提交”“下一个”等按钮）可以验证并处理数据。很多情况下，该模式与 Drop-Down List 或其他模式（例如，SEARCH<drop down list>FOR<fill-in-blanks textbox>）结合起来使用。
- 导航元素：文本或图形指示器启动验证并处理数据。

Navigation（导航）。在层次菜单、网页及交互式显示屏中为用户提供导航。

- 模式：Edit-In-Place。
- 概述：对于显示出的某些类型的内容，该模式提供了简单的文本编辑功能，而不需要用户明确指定文本编辑功能或状态。
- 详细资料：用户在显示屏上看到需要修改的内容。用鼠标在显示的内容上双击，就是向系统表示希望编辑该内容。该内容会突出显示，表示处于可编辑状态，用户就

① 这里用的是一个扼要的模式模板。完整的模式描述（连同数十个其他模式）可以在 [Tid02] 和 [Wel01] 中找到。

可以进行适当的修改了。

- 导航元素：无。

E-commerce（电子商务）。针对网站，这些模式能实现电子商务应用的重复元素。

- 模式：**Shopping Cart**。
- 概述：列出了要购买的选项清单。
- 详细资料：列出项目、数量、产品代码、可供性（有现货、无现货）、价格、交付信息、运费及其他相关的购买信息。同时也提供了编辑功能（例如，删除、修改数量）。
- 导航元素：包括继续购买或前往结账。

前面提到的每个模式的例子（每一类的所有模式）都有一个完整的构件级设计，包括设计类、属性、操作以及界面等。

关于用户界面模式的全面讨论超出了本书的范围。如果想进一步了解，请参看 [Yah13]、[UXM10]、[Gub09]、[Duy02]、[Tid02] 及 [Bor01]。

16.6 WebApp 设计模式

通过这一章的学习，我们了解到模式具有不同的类型及不同的分类方法。在建立 WebApp 的过程中，考虑必须要解决的设计问题时，侧重于二维的模式分类值得考虑，即以模式为焦点的设计以及粒度的级别。设计焦点标识着设计模型的哪个方面是相关的（例如，信息体系结构、导航、交互作用）。粒度则标识着要考虑的抽象级别（例如，该模式是否适用于整个 WebApp、单个网页、子系统或单独的 WebApp 构件）。

[364]

16.6.1 设计焦点

在前面几章中，强调了设计过程首先要从考虑体系结构、构件级问题及用户界面表示开始。随着设计的深入，设计的焦点就会变得更“窄”。在为 WebApp 设计信息体系结构时要遇到的问题（解决方案）与在执行界面设计时遇到的问题（解决方案）是不同的。因此，针对不同级别的设计焦点来开发 WebApp 设计模式并不奇怪，这样可以处理在每个抽象级别上遇到的独特问题（及相关的解决方案）。可按以下设计焦点的级别对 WebApp 模式进行分类：

- **信息体系结构模式**涉及信息空间的总体结构及用户与信息进行交互的方式。
- **导航模式**定义导航链接结构，例如层次、环、导航路径等。
- **交互模式**解决界面如何将特定动作的结果通知给用户。
- **表示模式**处理如何组织用户界面控制功能以获得更好的可用性，如何表现界面动作与所影响的内容对象间的关系，以及如何建立有效的内容层次。
- **功能模式**定义了 workflow、行为、过程、通信以及 WebApp 中的其他算法元素。

大多数情况下，在交互设计中遇到问题时，去研究信息体系结构模式的集合是白费力气的。这时应该去研究交互模式，因为交互模式才是与目前工作有关的设计焦点。

16.6.2 设计粒度

当我们要解决的问题涉及“全局”时，可以试着开发关注全局的解决方案（并使用相关模式）。相反，当焦点很窄时（例如，从包含五项或更少元素的集合中选择一项），解决方案（及相应模式）的目标也很窄。根据粒度的级别，WebApp 模式遵循本章前面所讨论的相同的抽象层次。

[365]

体系结构模式定义 WebApp 的总体结构,表示不同构件或增量间的关系,为指定体系结构元素(网页、包、构件、子系统)间的关系定义规则。设计模式处理特定的 WebApp 设计元素(例如构件的聚合)来解决一些设计问题、网页上元素间的关系或影响构件间通信的机制。构件模式与 WebApp 的个别小规模元素有关。这方面的例子有:个别交互元素(单选按钮,文本框),导航元素(如何格式化链接)或功能性元素(特定算法)。

信息栏 超媒体设计模式资源库

在互联网上有一些有用的超媒体模式目录及资源库,在这些代表性网站中描述了数以百计的设计模式:

- Tom Erickson 提出的交互模式 (www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html)。
- Martijn van Welie 提出的 Web 设计模式 (www.welie.com/patterns/)。
- 用于 UI 设计的 Web 模式 (<http://www.onextrapixel.com/2010/11/03/15-ui-design-patternsweb-designers-should-keep-handly/>)。
- 用导航模式改善 Web 信息系统 (<http://www8.org/w8-papers/5b-hypertext-media/improving/improving.html>)。
- 用户界面和 Uzilla.net 相关模式编译 (http://uzilla.net/uzilla/blog/hci_directory/searchresultde45.html)
- Common Ground——用于 HCI 设计的模式语言 (www.mit.edu/~jtidwell/interaction_patterns.html)。
- 个人网站的模式 (www.rdrop.com/~half/Creations/Writings/Web.patterns/index.html)。

16.7 移动 App 模式

从本质上说,移动 App 都是关于界面的。在许多情况下,移动用户界面模式 [Mob12] 在各种不同的应用程序类别中表示为一组“最佳的”屏幕图像。典型的例子可能包括如下内容。

登录屏幕。如何从一个特定的位置登录到屏幕,检查评论,并在社交网络中与朋友和粉丝分享评论?

地图。如何显示出一个在当前环境中能够解决一些其他问题的应用地图?例如,查找一个餐厅并且在市区中定位它的位置。

漂浮选单。如何表示实时产生或作为用户操作结果的一条消息或信息(来自应用程序或另一个用户)?

注册流程。如何提供一种简单的方式来登录或注册信息或功能?

自定义选项卡导航。如何表示出各种不同的内容对象,使用户可以选择一个自己想要的对象?

邀请。如何告诉他必须参与一些活动或对话?典型的例子如图 16-3。

有关移动用户界面模式的更多信息可参见 [Nei12] 和 [Hoo12]。除了用户界面模式之外,Meier 和他的同事还为移动 App 提出了许多更普遍的模式描述。关于移动模式的更多信息,包括一个广泛的模式库^①已经由 Nokia[Nok13] 发展起来。

① 各种移动模式库的新观点可以在 <http://4ourth.com/wiki/Other%20Mobile%20Pattern%20Libraries> 找到。

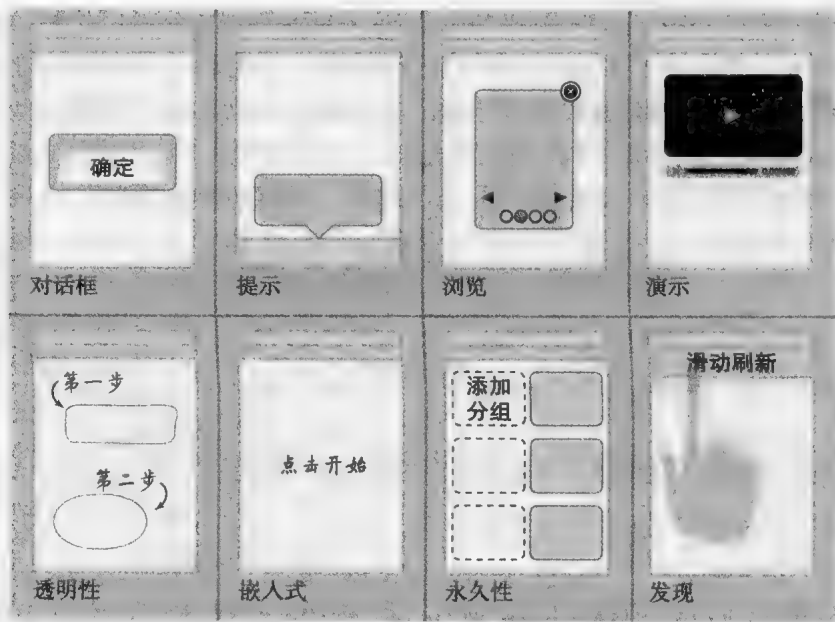


图 16-3 移动邀请模式示例 [Nei11]

16.8 小结

设计模式为描述问题及其解决方案提供了一种机制，允许软件工程组织获取可重用的设计知识。一个模式描述了一个问题，使用户能够理解问题所处的环境，并列出了影响因素，用来表明在环境中是如何解释这个问题的，以及如何应用解决方案的。在软件工程的工作中，我们识别有生产力的模式，并将其制成文档，该模式用来描述一个系统的重要且可重复的方面，然后在给定环境是唯一的影响因素时，提供一种方式来构建这个可重复的方面。

[367]

体系结构模式描述了广泛的设计问题，这些问题是用结构化方法来解决的。数据模式描述了面向递归数据的问题以及解决这些问题的数据模型解决方案。构件模式（也称设计模式）解决与子系统和构件开发相关的问题，还有相互之间通信的方式，以及它们在一个较大的体系结构中的位置。界面设计模式描述常见的用户界面问题以及在影响因素（包括最终用户的具体特征）内的解决方案。WebApp 模式解决在建立 WebApp 时遇到的问题集，通常与刚才提到的很多其他模式类别结合使用。移动模式解决了移动界面的独特性质和功能以及移动平台特有的控制元素。

框架提供了模式所在的基础结构，而习惯用语描述了特定程序设计语言对于全部或部分指定算法和数据结构的实现细节。标准化的表格或模板用来描述模式。模式语言包含模式集，每个模式都使用一个标准化的模板来描述，这些模式之间相互关联来显示它们如何协作来解决应用领域中的问题。

基于模式的设计与体系结构、构件级以及用户界面设计方法联合使用。这种设计方法从检查需求模型开始，分解问题、定义环境并描述影响因素。接下来，查找问题领域的模式语言，以确定已被分解问题的模式是否存在。一旦找到了适合的模式，就可用作设计指南。

习题与思考题

16.1 讨论设计模式的三个“部分”，对于每个部分从软件以外的其他领域提供具体的例子。

- 16.2 无生产力模式和有生产力模式之间的区别是什么?
- 16.3 体系结构模式与构件级模式有什么区别?
- 16.4 什么是框架? 框架与模式的区别在哪里? 什么是习惯用语? 习惯用语与模式的区别是什么?
- 16.5 用 16.1.3 节描述的设计模式模板, 为老师提出的一个模式开发完整的模式描述。
- 16.6 为你熟悉的一项运动开发一个简要模式语言。你可以从处理环境、系统影响因素、教练和球队必须解决的主要问题开始。你只需要指定模式名称, 并用一句话来描述每个模式。 368
- 16.7 寻找 5 个模式资源库, 并且简要描述包含在其中的模式类型。
- 16.8 Christopher Alexander 说, 简单地通过把可执行部分加在一起, 不能实现好的设计。你认为他说的是什么意思?
- 16.9 用 16.2.3 节提到的基于模式的设计任务为 15.3.2 节中的“室内设计系统”开发一个简要设计。
- 16.10 为在习题 16.9 中用到的模式建立一个基于模式的组织表。
- 16.11 用 16.1.3 节介绍的设计模式模板为 16.3 节提到的厨房模式开发一个完整的模式描述。
- 16.12 “四人帮”[Gam95] 提出了很多构件模式, 适用于面向对象系统。从中选择一个(可以在网上找到)进行讨论。
- 16.13 对于用户界面模式, 查找三个模式资源库。从每个资源库中选择一个模式进行简要描述。
- 16.14 对于 WebApp 模式, 查找三个模式资源库。从每个资源库中选择一个模式进行简要描述。
- 16.15 对于移动模式, 查找三个模式资源库。从每个资源库中选择一个模式进行简要描述。

扩展阅读与信息资源

在基于模式的设计方面, 有很多书籍可供软件工程师选择。Gamma 和他的同事 [Gam95] 曾经编写了一本关于这个主题的开创性的书籍。最近出版的书籍包括: Burris (《Programming in the Large with Design Patterns》, Pretty Print Press, 2012)、Smith (《Elemental Design Patterns》, Addison-Wesley, 2012)、Lasater (《Design Patterns》, Wordware Publishing, 2007)、Holzner (《Design Patterns for Dummies》, For Dummies, 2006)、Freeman 和她的同事 (《Head First Design Patterns》, O'Reilly Media, 2005) 以及 Shalloway 和 Trott (《Design Patterns Explained》, 2nd.ed., Addison-Wesley, 2004)。Kent Beck (《Implementation Patterns》, Addison-Wesley, 2008) 阐述了在构造活动中遇到的编码及实现方面的模式。

其他书籍主要关注特定应用开发和语言环境下的设计模式, 包括: Bowers 和他的同事 (《Pro HTML5 and CSS3 Design Patterns》, Apress, 2011)、Scott 和 Neil (《Designing Web Interfaces: Principles and Patterns for Rich Interactions》, O'Reilly, 2009)、Tropashko 和 Burleson (《SQL Design Patterns: Expert Guide to SQL Programming》, Rampant Techpress, 2007)、Mahemoff (《Ajax Design Patterns》, O'Reilly Media, 2006)、Bevis (《Java Design Pattern Essentials》, Ability First Limited, 2010)、Metsker 和 Wake (《Design Patterns in Java》, Addison-Wesley, 2006)、Millett (《Professional ASP.NET Design Patterns》, Wrox, 2010)、Nilsson (《Applying Domain-Driven Design and Patterns: With Examples in C# and .NET》, Addison-Wesley, 2006)、Stefanov (《JavaScript Patterns》, O'Reilly, 2010)、Sweat (《PHP | Architect's Guide to PHP Design Patterns》, Marco Tabini & Associates, 2005)、Paul (《Design Patterns in C#》)、Metsker (《Design Patterns C#》, Addison-Wesley, 2004)、Grand 和 Merrill (《Visual Basic.NET Design Patterns》, Wiley, 2003)、Crawford 和 Kaplan (《J2EE Design Patterns》, O'Reilly Media, 2003)、Juric 等人 (《J2EE Design Patterns Applied》, Wrox Press, 2002) 以及 Marinescu 和 Roman (《EJB Design Patterns》, Wiley, 2002)。

还有其他书籍涉及了特殊的应用领域, 包括: Kuchana (《Software Architecture Design Patterns in

Java 》, Auerbach, 2004)、Joshi (《 C++ Design Patterns and Derivatives Pricing 》, Cambridge University Press, 2004)、Douglass (《 Real-Time Design Patterns 》, Addison-Wesley, 2002) 以及 Schmidt 和 Rising (《 Design Patterns in Communication Software 》, Cambridge University Press, 2001)。

对于每个希望全面了解设计模式的软件设计师, 架构师 Christopher Alexander 所编写的经典著作 (《 Notes on the Synthesis of Form 》, Harvard University Press, 1964) 和 (《 A Pattern Language : Towns, Buildings, Construction 》, Oxford University Press, 1977) 很值得一读。

在网上可以获得大量基于模式的设计方面的信息资源。与基于模式的设计有关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

WebApp 设计

要点浏览

概念: WebApp 设计包括的技术性活动和非技术性活动有: 建立 WebApp 的外观和印象, 创建用户界面的美学布局, 定义总体结构, 开发体系结构中的内容和功能, 以及设计 WebApp 的导航等。

人员: Web 工程师、美术设计师、内容开发者及其他利益相关者都参加 Web 工程设计模型的创建。

重要性: 设计工作允许 Web 工程师创建模型, 可以对该模型进行质量评估, 并且可以在内容和编码生成之前、在测试开始之前以及在最终用户参与之前对该模型进行改进。通过设计达到 WebApp 质量要求。

步骤: WebApp 设计包括 6 个主要步骤, 这些步骤由分析建模阶段所获取的信息驱动。内容设计利用内容模型(在分析期

间开发的)作为建立内容对象设计的基础。美学设计(也称为平面设计)建立了最终用户所关注的外观和感觉; 结构设计重点关注所有内容对象和功能的总体超媒体结构; 界面设计创建了定义用户界面的总体布局和交互机制; 导航设计定义了最终用户是怎样对超媒体结构进行导航的; 构件设计体现了 WebApp 功能元素的详细内部结构。

工作产品: 设计模型包括内容、美学、体系结构、界面、导航及构件级设计问题, 它是 Web 工程设计的主要工作产品。

质量保证措施: 要对设计模型的每个元素进行评估, 尽力发现错误、不一致或者遗漏的地方。另外, 考虑可选的解决方案, 并对当前设计模型有效实现的程度进行评估。

Jakob Nielsen[Nie00]在他的有关 Web 设计的权威著作中这样写道: “本质上, 设计有两种基本途径: 表达你自己的艺术设想和为客户解决问题的工程设想。”在 Web 发展的前 10 年, 艺术设想是很多开发者选择的方式。他们以一种特别的方式进行设计, 并且通常是在生成 HTML 时才进行设计。设计从艺术想象力发展而来, 而艺术想象力本身又是随着 WebApp 结构的出现而发展的。

即使在今天, 还有很多 Web 开发者使用 WebApp 向孩子们宣传“有限设计”。他们认为 WebApp 的直接性和易变性削弱了形式化设计, 即设计是随着对应用系统进行构造(编码)而进化的, 并且应该花费较少的时间创建详细设计模型。这种论述有其优点, 但是只适用于相对简单的 WebApp。当内容和功能变得复杂时, 当 WebApp 的规模包含成百上千的内容对象、函数和分析类时, 当 WebApp 的成功对于业务成功具有直接影响时, 就不能轻视设计, 也不该轻视设计。

关键概念

美学设计
体系结构设计
构件级设计
内容体系结构
内容设计
内容对象
设计目标
设计金字塔
设计质量
平面设计
模式—视图—
控制器
导航设计
质量检查单
WebApp 体系
结构

这种情况将使我们考虑 Nielsen 提出的第二种途径——“为客户解决实际问题的工程设想”。Web 工程^①采纳了这一思想，一种更严格的 WebApp 设计方法使开发人员能够实现这种设想。

17.1 WebApp 设计质量

每个上网的人都对什么是“好的”WebApp 有自己的看法，大家看待这一问题的角度也相差甚远。有些用户喜欢闪烁的图示，有些人则喜欢简单的文本；有些用户想看到丰富的内容，而有些人只是渴望看到简略的陈述；有些人喜欢高级的分析工具或者数据库访问，而有些人只是需要一些简单应用。实际上，比起从技术角度讨论 WebApp 的质量，用户对于“好”的理解（基于这种理解而接受或拒绝 WebApp）可能更重要。

但是如何认识 WebApp 的质量呢？具有哪些特性的 WebApp 才能得到最终用户的好评？同时，在质量方面，具有哪些技术特点才能使 Web 工程师可以长期对 WebApp 进行修正性维护、适应性维护、增强性维护及支持？

实际上，在第 12 章讨论的软件质量的所有技术特征，以及第 19 章介绍的通用质量属性都适用于 WebApp。然而，其中一些最相关的通用特性——可用性、功能性、可靠性、效率及可维护性——为评估基于 Web 的系统的质量提供了有用基础。

Olsina 和他的同事 [Ols99] 设计了一个“质量需求树”，定义了一组可产生高质量 WebApp 的技术属性，包括可用性、功能性、可靠性、效率和可维护性^②。图 17-1 总结了他们的工作。在图中提及的标准对于长期从事设计、构造和维护 WebApp 的工程师是有帮助的。

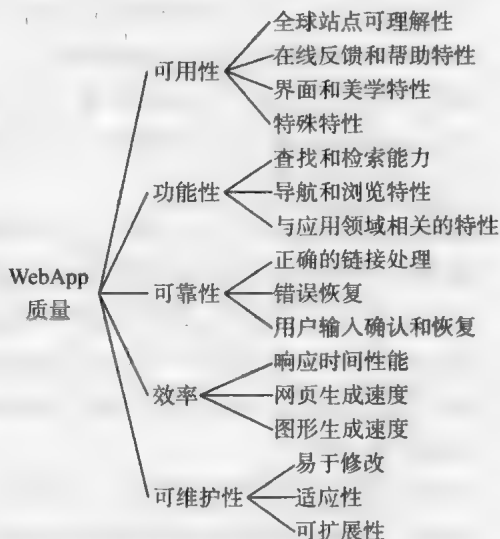


图 17-1 质量需求树 [Ols99]

Offutt[Off02] 又补充了下面的属性，对图 17-1 描述的 5 个质量属性进行了扩展。

[372]

安全性。WebApp 已经和重要的公司及政府数据库高度集成。电子商务应用系统提取敏感的客户信息，然后将这些信息存储起来。由于这些及许多其他原因，WebApp 的安全性在很多情况下变得极为重要。安全性的关键度量标准是 WebApp 和服务环境拒绝非授权访问和阻挡恶意攻击的能力。在第 27 章进行了安全工程的讨论。关于 WebApp 的更多信息可以参考 [Web13]、[Pri10]、[Vac06] 和 [Kiz05]。

可用性。如果不可用的话，即使是最好的 WebApp 也不能满足用户的要求。从技术的角度来看，可用性是对 WebApp 的可用时间占总时间的百分比的一种度量。但是 Offutt[Off02] 认为，“使用仅限于在一种浏览器或平台上可用的特性”会使 WebApp 在那些具有不同浏览

提问 WebApp 的主要质量属性是什么？

① Web 工程 [Pre08] 是本书介绍的软件工程方法的改编版。为了建立达到工业质量要求的基于 Web 的系统和应用，人们提出了一种敏捷但严格的框架。

② 这些质量属性与在第 12 章和第 19 章介绍的属性十分相似。这意味着，质量特征对所有的软件都是通用的。

器或平台的配置中变得不可用，用户会毫无例外地转向其他地方。

可伸缩性。WebApp 和其服务环境能否扩展来处理 100 个、1000 个、10000 个或 100000 个用户？WebApp 和为其提供接口的系统能不能承受访问数量上的巨大波动？响应速度是否会因此而剧减（或者完全停止）？设计能够成功调节负载（例如，相当多的最终用户）的 WebApp 同样重要，而且会变得越来越重要。

投放市场时间。虽然投放市场时间并不是真正的技术方面的质量属性，仅仅是从商业角度考虑的一种质量度量。但是，市场上的第一个 WebApp 往往能够吸引非常多的最终用户。

373

信息栏 WebApp 设计——质量检查单

下面的清单是根据 Webreference.com 上所提供的信息修改而来的，清单中列出了一组问题来帮助 Web 工程师和最终用户评估总体的 WebApp 质量。

- 内容、功能导航选项能否按照用户的喜好而定制？
- 内容和功能能否按照用户通信所用的带宽进行定制？
- 图形和其他非文本媒体能否正确使用？是否出于显示效率方面的考虑而对图形文件的大小进行了优化？

- 是否用可以理解的、有效显示的方式来组织表格，并按大小进行排序？
- 是否对 HTML 进行优化来消除低效率？
- 总体页面设计是否容易阅读和导航？
- 是否所有的指针都提供了指向用户感兴趣信息的链接？
- 是否大部分的链接在 Web 中都具有持久性？
- WebApp 是否提供了站点管理工具？包括使用跟踪、链接测试、本地搜索和安全性工具。

在 WWW 上查找信息的人可以获得数亿的网页。即使是很好地命中目标的 Web 查找也会得到大量内容。要从这么多的信息源中选择需要的信息，用户如何评价 WebApp 所展示内容的质量（例如，准确性、精确性、完整性、适时性）呢？

Tillman[Til00] 提出了评价内容质量的一组有用标准：能否很容易地判断内容的范围和深度，确保满足用户的要求？是否容易识别内容作者的背景和权威性？能否确定内容的通用性？最后的更新时间及更新内容是什么？内容和位置是否稳定（即它是否一直保存在引用的 URL 处）？内容是否可信？内容是否独特？也就是说，WebApp 能否给使用它的用户带来一些特别的好处？内容对于目标用户群体是否有价值？内容的组织是否合理？是否有索引？是否容易选取？这些问题只是设计 WebApp 时应该考虑的问题中的一小部分。

提问 评估内容质量时我们应该考虑什么？

17.2 设计目标

在 Web 设计的定期专栏中，Jean Kaiser[Kai02] 提出了下面的设计目标，无论应用的领域、规模和复杂度如何，这些目标实际上可以适用于任何 WebApp。

简单性。虽然可能有些过时，但是格言“一切都要适度”也适用于 WebApp。设计者倾向于给用户“太多的东西”——详尽的内容、完美的视觉效果、插入的动画、大量的网页、复杂的导航等，但是最好还是尽量做到适度和简单。

引述 你能做并不意味着你应该这么做。

Jean Kaiser

374

一致性。这一设计目标几乎适用于设计模型的每个元素。内容构造应该一致（例如，在所有相关的文档文件中，文本格式和字体风格都应该保持一致；图形应该有统一的外观、颜色配置和风格）。美术设计（美学方面）应该在 WebApp 的各部分有统一的外观。应该在所有的 WebApp 元素中一致地使用导航机制。如 Kaiser[Kai02] 所提到的：“记住，对于访问者来说，一个网站是一个物理位置。如果网站的网页在设计上不一致，会使人感到困惑。”

符合性。WebApp 的美学、界面和导航设计必须与将要构造的应用所处的领域保持一致。毫无疑问，音乐公司的网站与提供财务服务的公司主页在外观和感觉上肯定不同。

健壮性。在已经建立的符合性的基础上，WebApp 通常会给用户以不言而喻的“承诺”。用户期待与他们的要求相关的健壮的内容和功能，如果这些元素遗漏或不足，WebApp 很可能会失败。

导航性。导航性应该以直观的和可预测的方式来设计。也就是说，用户不必搜索导航链接和帮助就知道如何使用 WebApp。例如，选作导航机制的图标或图像必须能够直观地识别。在很多图形图像中寻找可用的链接是很令人厌烦的。

视觉吸引力。在所有类型的软件中，WebApp 毫无疑问是最具有视觉效果、最生动的，也是最具有审美感的。美丽的外观（视觉吸引力）无疑是最吸引观看者眼球的，然而许多设计特性（例如，内容的外观、界面设计、颜色协调、正文布局匀称、图片和其他媒体、导航机制）也会对视觉吸引力产生影响。

兼容性。WebApp 会应用于不同的环境（例如，不同的硬件、Internet 连接类型、操作系统、浏览器），并且必须设计为互相兼容。

引述 对于有些人来说，Web 设计注重视觉效果……而对于另外一些人来说，Web 设计是通过文档空间来构造信息和导航。还有些人可能将 Web 设计看成是一种用于构建交互式 WebApp 的技术。实际上，设计应该包括所有这些内容，并且可能比这些还多。

Thomas Powell

17.3 WebApp 设计金字塔

在 Web 工程的背景下，什么是设计？这个问题可能比想象的更难于回答。Pressman 和 Lowe [Pre08] 是这样讨论这个问题的：

创建有效的设计通常需要各种技术。有时，对于小项目，开发者可能需要成为多面手。对于比较大的项目，借鉴专家的专业知识是明智且可行的，例如 Web 工程师、平面设计师、内容开发者、程序员、数据库专家、信息架构师、网络工程师、安全专家及测试人员。借鉴这些技术，可以评估创建的模型质量，在产生内容和代码、进行测试以及大量的终端用户参与之前，对模型进行改进。如果分析是建立 WebApp 质量，那么设计就是真正地嵌入质量。

要根据 WebApp 性质的不同，而适当混合各种技术。图 17-2 描述了 Web 工程的设计金字塔，金字塔的每一层都表示一种设计动作，这些设计动作在后面的章节中介绍。

引述 如果一个站点非常好用，但却缺乏美感和合适的设计风格，它也同样会失败。

Curt Cloninger



图 17-2 WebApp 设计金字塔

375

17.4 WebApp 界面设计

当用户与基于计算机的系统交互时，要应用一套基本原则和重要的设计准则。这些已在

第 15 章讨论^①。虽然 WebApp 提出了一些特殊的用户界面设计上的挑战，但是基本原则和方针仍然是适用的。

WebApp 界面设计的挑战之一是，用户的进入点不明确。即用户可能在主页进入 WebApp，或者可能链接到 WebApp 体系结构的一些较低层。在某些情况下，可用更改用户到主页的路线方法来设计 WebApp，但是如果不想这样做，那么 WebApp 设计必须提供界面导航特征以及全部内容对象，这样就可以不考虑用户是如何进入系统的。

WebApp 界面的目标是：（1）建立一致性的窗口，用户由此进入界面提供的内容和功能；（2）通过一系列与 WebApp 的交互指导用户；（3）组织用户可用的导航选项和内容。为了获得一致的界面，首先要用美学设计（17.5 节）去建立一致的外观。这包括很多特点，但必须强调布局和导航机制的形式。为了指导用户交互，可以适当借鉴隐喻（metaphor）^②，使用户能直观地理解界面。为了实现导航选项，可以选择网页中位置固定的导航菜单，可以选择使用户识别为导航元素的图标，也可以选择链接到内容主题或 WebApp 功能的图像。值得注意的是，在内容层次的每个级别上都应提供一种或多种导航机制。

建议 不是每位 Web 工程师（或软件工程师）都有艺术（美学）才能。如果你也没有这种才能，那么就雇用有一个有经验的平面设计师进行美学设计工作。

376

17.5 美学设计

美学设计，又称平面设计，是一种艺术工作。它是对 Web 设计在技术方面的补充。没有它，WebApp 可能有强大的功能，但是却不能吸引人；有了它，WebApp 能够将它的用户带入以用户为核心的充满智慧的世界。

但是什么是美学？有这么一句谚语：“美丽存在于能够发现美丽的人的眼中。”当考虑 WebApp 的美学设计时，这句话就更加贴切了。为了进行有效的美学设计，我们再一次回到作为分析模型的一部分而开发的用户层次（第 8 章），并且提出这样的问题：“谁是 WebApp 的用户，他们想要什么样的‘外观’？”

提问 对于 WebApp 设计者来说，可以使用哪些交互机制？

SafeHome 平面设计

[场景] Doug Miller 的办公室，第一个 Web 界面原型评审后。

[人物] Doug Miller，软件工程项目经理；Vinod Raman，SafeHome 软件工程团队成员。

[对话]

Doug：你对新的 Web 页面设计的印象如何？

Vinod：我喜欢它，但更重要的是，我们的客户喜欢。

Doug：你从我们市场营销部聘来的美术设计师那里得到了多少帮助？

Vinod：确实很多。她对页面布局有很好的眼光，并且提出了一个极好的页面图形主题布局，比我们自己提出的方案好多了。

Doug：很好。还有其他问题吗？

① 15.5 节专门介绍了 WebApp 界面设计。如果还没有读过，现在可以去读。

② 在本书中，隐喻是一种表示方式（来自用户工作经验），可以在界面环境内建模。一个简单的例子是控制 .mpg 文件音量的滚动条开关。

Vinod：考虑到一些视觉上受损用户的访问性问题，我们还需要创建帮助页面。但是这样的话，我们将不得不为所有设计的页面考虑这个问题。

Vinod：当然。这位设计师对可用性和可访问性问题有很好的理解。

Doug：好吧，我问一下市场营销部我们是否能聘用她长一点儿的时间。

Doug：我们还需要平面设计的帮助吗？

377

17.5.1 布局问题

每个网页中能够用来支持非功能性的美学设计、导航特征、信息内容及指导用户功能的“空间”都是有限的，应该在美学设计期间对这种空间的“开发”进行规划。

像所有的美学问题一样，设计屏幕布局也没有绝对的规则。但是，很多一般的布局指导原则还是值得考虑的。

不要担心留下空白空间。我们不建议把网页中的每一寸空间都排满信息。如果非要这样做，用户寻找有用信息或要素会很困难，并会造成很不舒服的视觉混乱。

重视内容。毕竟，内容是用户浏览网页的根本原因。Nielsen[Nie00]建议，典型的 Web 网页应用的 80% 应该是内容，剩余的资源为导航和其他要素。

按照从左上到右下的顺序组织布局元素。绝大多数用户浏览网页的方式与看书没有什么不同——从左上到右下^①。如果布局元素有特定的优先级，应该将高优先级的元素放在页面空间的左上部分。

在页面内按导航、内容和功能安排布局。在几乎所有的事情中，人们都会寻找事实上已有的模式。如果 Web 页中没有可辨别的模式，用户的挫败感会增加（由于需要对所需要的信息进行不必要的查找）。

不要通过滚动条扩展空间。虽然滚动经常是需要的，但大多数的研究表明，用户还是不喜欢用滚动条。通常最好是减少网页内容，或者通过多页显示必要的内容。

在设计布局时，考虑分辨率和浏览器窗口的尺寸。设计应该能够确定布局元素占用可用空间的百分比，而不是在布局中规定固定的尺寸[Nie00]。随着越来越多的具有不同屏幕尺寸移动设备的使用，这个概念变得越来越重要。

17.5.2 平面设计问题

平面设计需要考虑到 WebApp 外观的每个方面。平面设计过程从布局（17.5.1 节）开始，并在设计过程中考虑全局颜色配置、字体、字号、风格、补充媒体（例如，音频、视频、动画）的使用，以及应用中的所有其他美学元素。

关于 WebApp 的平面设计问题的全面讨论超出了本书的范围。感兴趣的读者能够从很多专业网站（例如，www.graphic-design.com、www.webdesignfromscratch.com、www.wpdtd.com）或一些可打印的资源（例如，[Bea11]、[McN10]、[Lup08] 以及 [Roc06]）中获得设计忠告和指导原则。

引述 我们发现人们仅仅通过视觉设计的效果来快速评估一个网站。
指导 Web 可信性设计的 Stanford 指导原则

① 基于文化和语言的不同，也有例外，但这条规则对多数用户来说是有效的。

378

信息栏 设计良好的网站

有时候，要理解好的 WebApp 设计，最好的方法就是看几个例子。在 Marcelle Toor 的文章“The Top Twenty Web Design Tips”中 (<http://www.graphic-design.com/Web/feature/tips.html>)，他建议将下面的网站作为平面设计的范例。

- <http://www.marcelletoordesigns.com/>。ILR 网站设计为网站开发人员和其他互联网企业提供了前沿资源。
- www.workbook.com。这个网站展示了插图画家和设计者的工作。
- www.pbs.org/riverofsong。针对公众电视的电视连续剧和关于美国音乐的无线电广播。
- www.RKDINC.com。提供在线代表作品选和设计技巧的设计公司。
- <http://www.creativehotlist.com/>。介绍了由广告代理商、平面设计公司以及其他通信专家开发的设计良好的网站。
- www.btdnyc.com。Beth Toudreau 领导的设计公司。

17.6 内容设计

内容设计关注两个不同的设计任务，每个人都可以使用不同的技巧描述每个任务。首先，需要为内容对象开发一种设计表示，并且开发一种机制建立内容对象之间的关系。其次，要生成特定的内容对象内的信息。后者可由广告撰稿人员、平面设计人员以及设计 WebApp 内容的其他人员完成。

17.6.1 内容对象

在 WebApp 设计环境中，内容对象与传统软件中的数据对象关系更加紧密。内容对象具有的属性，包括特定的内容信息（通常在 WebApp 需求建模期间定义）的属性和指定为设计成分的实现属性。

举个例子，考虑为 SafeHome 电子商务系统开发的分析类 ProductComponent。分析类的属性 description 在这里被描述为一个设计类，名为 CompDescription。这个类包括 5 个内容对象：MarketingDescription、Photograph、TechDescription、Schematic 和 Video，如图中 17-3 中阴影部分所示。内容对象所包含的信息被标注成对象的属性。例如，Photograph（一个 .jpg 格式的图示）包含属性 horizontal dimension、vertical dimension 和 border style。

UML 的关联和聚合符号^①可以用来表示内容对象之间的关系。例如，图 17-3 所示的 UML 关联表明一个 CompDescription 类对象用于描述一个 ProductComponent 类实例；一个 CompDescription 类实例由所示的 5 个内容对象组成。然而，所示的多重性符号表明 Schematic 类实例和 Video 类实例是可选的（值可能为 0），一个 MarketingDescription 类实例和一个 TechDescription 类实例是必需的，会用到一个或多个 Photograph 类实例。

引述 好的设计人员能够使混乱的状态正常化，他们能够通过组织和管理文字和图片清楚地表达自己的想法。

Jeffery Veen

379

① UML 聚合和关联的表示在附录 1 中讨论。

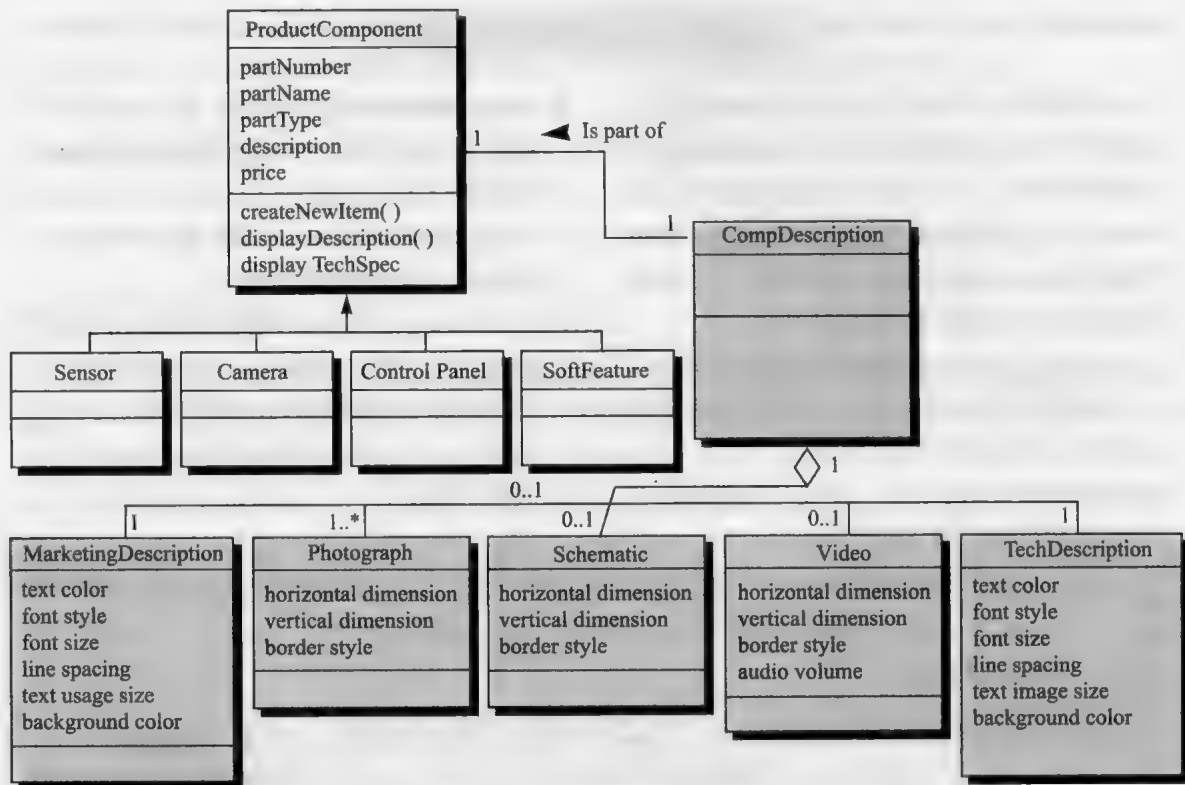


图 17-3 内容对象的设计表示

17.6.2 内容设计问题

对所有的内容对象建模之后，就必须编写对象传递的信息，然后对其格式化，最大程度地满足用户的要求。内容编辑是相关领域专家的工作，他们通过提供所传递信息的概要描述和用来传递信息的一般内容对象的类型说明（例如，描述性文本、图片、照片）来设计内容对象，可能也会应用美学设计（17.5 节）为内容设计合适的外观。

[380]

设计内容对象时，将内容对象“分块”[Pow02]，然后形成 WebApp 页面。集成在一个页面的内容对象的数量与用户需求、网络连接的下载速度以及用户能够忍受的滚动次数有关。

17.7 体系结构设计

体系结构设计与已建立的 WebApp 的目标、展示的内容、将要访问它的用户和已经建立的导航原则紧密相关。体系结构设计者必须确定内容体系结构和 WebApp 体系结构。内容体系结构^①着重于内容对象（诸如网页的组成对象）的表现和导航的组织方式。WebApp 体系结构描述应用将以什么组织方式来管理用户交互、操纵内部处理任务、实现导航及展示内容。

在大多数情况下，体系结构设计与界面设计、美学设计和内容设计并行进行。由于 WebApp 的体系结构对导航的影响很大，所以在设计活动中

引述 相对于水平滚动，用户往往更加倾向于容忍垂直滚动。因此要避免宽页面格式。

设计良好的网站，其体系结构不总是对用户透明的——也不应该这样。

Thomas Powell

① 术语信息体系结构也用于表示结构，使得我们能够更好地组织、标识、导航及搜索内容对象。

做出的决定会影响导航设计阶段的工作。

17.7.1 内容体系结构

内容体系结构的设计着重于对 WebApp 的所有超媒体结构进行定义。虽然有时可以创建定制的体系结构，但通常会在下面四种不同的内容结构中进行选择 [Pow02]。

线性结构。当内部交互的可预测顺序（带有一些变化或转向）很常见时，便会遇到线性结构（图 17-4）。帮助文档的展示可能是一个典型的例子，在有必备的前提信息后，信息页连同相关的图示、简短的视频、音频才能随之出现。内容展示的顺序是预先定义的，而且通常是线性的。还有一个例子是产品订单的输入顺序，必须以特殊的顺序对特殊的信息进行详细说明。在这种情况下，图 17-4 所示的结构是很合适的。当内容和处理过程变得越来越复杂时，图左边所示的纯粹的线性流程将被更复杂的线性结构所取代，在此结构中，可替换的内容可以被触发或者发生转向来获取补充的内容（图 17-4 右边所示的结构）。

提问 一般会遇到哪些类型的内
容体系结构？

381

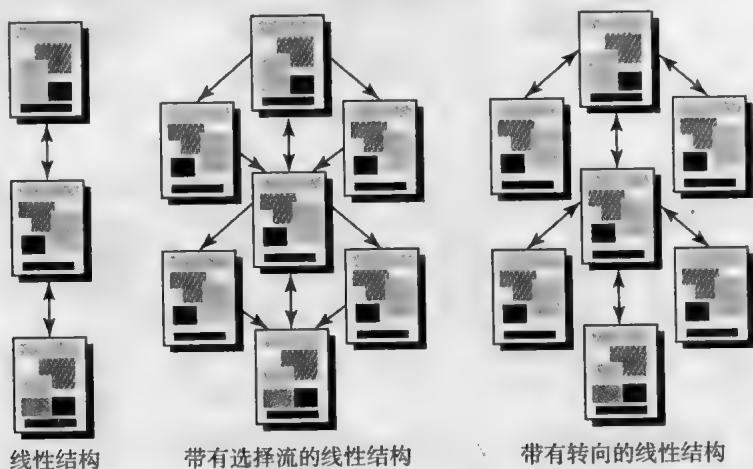


图 17-4 线性结构

网格结构。网格结构（图 17-5）是另一种体系结构，当 WebApp 的内容按类别组织成二维（或更多维）时，可以采用这种结构。例如，考虑这样的情况：一个电子商务网站销售高尔夫球棒。网格的水平方向代表要出售的球棒种类（例如木制棒、金属棒、楔形棒、轻击棒）；垂直方向代表不同的球棒制造厂商所提供的产品。因此，用户可能沿着网格的水平方向找到轻击棒所在的列，然后在竖直方向上检查销售轻击棒的厂家提供的产品。这种 WebApp 结构只有在内容十分规则的情况下才会使用 [Pow02]。

分层结构。分层结构（图 17-6）毫无疑问是最常见的 WebApp 体系结构。与第 13 章讨论的分区软件层次有所不同，在第 13 章，只在层次的垂直分支上支持控制流程，而 WebApp 的层次结构可以设计成使控制流水平地穿过垂直分支（通过超文本分支）的方式。

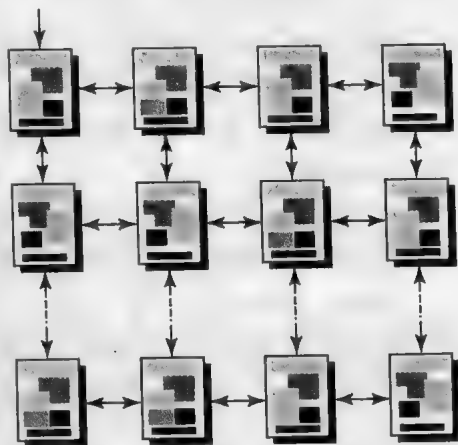


图 17-5 网格结构

382

因此,在分层结构中最左边展示的内容可以通过超文本链接与中间或者右边分支的内容相连。然而,我们应该注意,虽然这种分支结构能够实现 WebApp 内容的快速导航,但同时有可能给用户带来困惑。

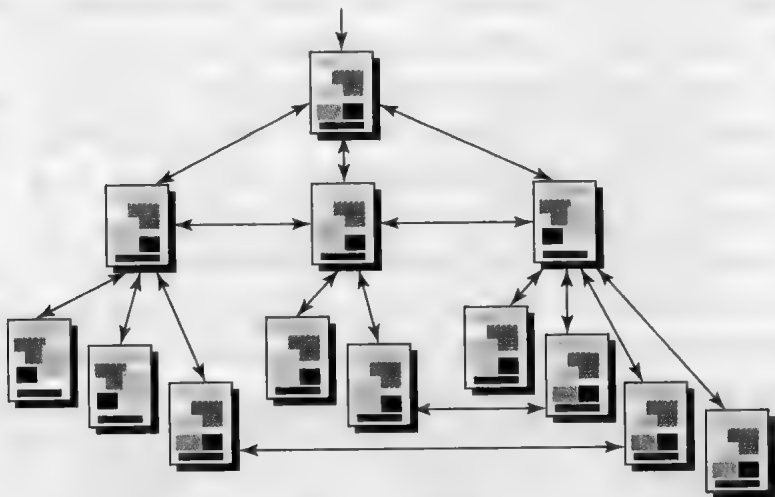


图 17-6 分层结构

网络结构或“纯 Web”结构。这种结构(图 17-7)在很多方面类似于为面向对象系统演化的体系结构。对结构构件(此处为网页)进行设计,使得这些构件能够将控制传递(通过超文本链接)到系统中的几乎所有其他构件。这种方法使导航相当灵活,但同时可能使用户感到困惑。

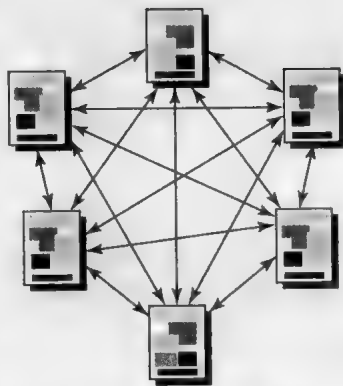


图 17-7 网络结构

可以将前面段落中所讨论的设计结构进行组合,形成复合结构。WebApp 的总体结构可能是层次结构,但是部分结构可能会展示出线性特性,而结构的另一部分可能是网络结构。结构设计人员的目标就是使 WebApp 的结构和展示的内容及实现的过程相匹配。

17.7.2 WebApp 体系结构

WebApp 体系结构描述了使基于 Web 的系统或应用达到其业务目标的基础结构。Jacyntho 和他的同事 [Jac02b] 对这一结构的基本特性做了如下描述:

创建应用程序应该考虑到不同层所关注方面的不同,特别是,应用系统数据应该与网页的内容(导航节点)分开,反过来,网页的内容也要与界面的外观(页面)清楚地分离。

作者建议采用三层设计结构,使界面与导航及应用行为相分离,并且认为使界面、应用和导航分离可以简化实现,并能够增加复用性。

模型-视图-控制器(Model-View-Controller, MVC)结构 [Kra88]^①是许多建议的 WebApp 基础结构模型之一,它将用户界面与 WebApp 功能及信息内容分离。模型(有时称“模型对象”)包括应用的所有详细内容和

关键点 MVC 体系结构将用户界面与 WebApp 功能和信息内容分离。

① 值得注意的是, MVC 实际上是为 Smalltalk 环境开发的体系结构设计模式(www.smalltalk.org),并且可用于任何交互式应用软件。

处理逻辑，还包括所有内容对象、对外部数据或信息源的访问，以及应用的特定处理功能；视图包括所有界面的特定功能，并能够表示内容和处理逻辑，包括所有内容对象、对外部数据或信息源的访问，以及最终用户所需要的所有处理功能；控制器管理对模型和视图的访问，并协调两者间的数据流。在 WebApp 中，“视图由控制器进行更新，更新数据来自基于用户输入的模型” [WMT02]。MVC 体系结构的示意图如图 17-8 所示。

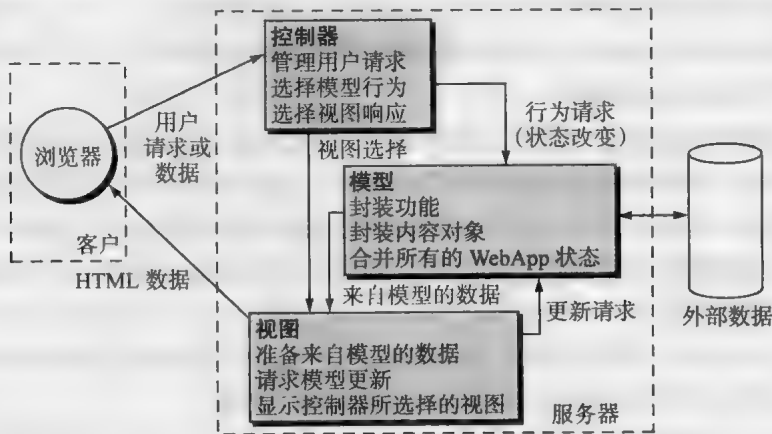


图 17-8 MVC 结构 (改自 [Jac02b])

根据此图，用户请求或数据由控制器处理。控制器也可以根据用户请求选择合适的视图对象。一旦确定了请求的类型，就将行为请求传递给模型，模型实现功能，或者检索满足用户请求所需要的内容。模型对象可以访问存储在公司数据库中的数据，被访问的数据可以与本地数据存储在一起，或者作为一些独立文件单独存储。模型创建的数据必须由合适的视图对象对其进行格式化和组织，然后从应用服务器传回到客户端浏览器，并显示在客户的计算机上。

384

很多情况下，在实现应用系统的开发环境中定义 WebApp 的体系结构。感兴趣的读者请查阅 [Fow03]，其中深入讨论了开发环境及其在 WebApp 体系结构设计中的作用。

17.8 导航设计

一旦建立了 WebApp 的体系结构并确定了体系结构的构件（页面、脚本、applet 和其他处理功能），设计人员就应定义导航路径，使用户可以访问 WebApp 的内容和功能。为了完成这一任务，要为网站的不同用户确定导航语义，并且定义实现导航的机制（语法）。

17.8.1 导航语义

像很多 WebApp 设计活动一样，在进行导航设计时，首先考虑用户层次和为每一类用户（角色）创建的相关用例（第 9 章）。每一类角色使用 WebApp 的方式或多或少会有所区别，因而会有不同的导航要求。另外，为每一类角色设计的用例会定义一组类，这组类包含一个或多个内容对象，或者包含 WebApp 功能。当用户与 WebApp 进行交互时，会接触到一系列的导航语义单元（Navigation Semantic Unit, NSU）——“信息和相关的导航结构的集合，它们相互协作共同完成相关的用户请求的一部分” [Cac02]。

引述 Gretel，等到月亮升起来的时候，我们就能看到我撒落的面包屑。这些面包屑会指引我们回家的路。

Hansel, Gretel

385

NSU 由一组导航元素组成, 其中, 导航元素也称作导航方式 (Way of Navigating, WoN) [Gna99]。对于特定类型的用户来说, 为了达到导航目的, WoN 展示了最佳的导航路径。每个 WoN 由一组相关的导航节点 (Navigational Node, NN) 组成, 这些导航节点通过导航链接连接起来, 在某些情况下, 导航链接可能就是另一个 NSU。因此, 可以将 WebApp 的总体导航结构组织为 NSU 的层次结构。

为了举例说明 NSU 的开发, 考虑选择 SafeHome 部件用例。

用例: 选择 SafeHome 部件

WebApp 会推荐产品部件 (例如, 控制面板、传感器、摄像机) 及每个房间和外部入口的其他特征 (例如, 用软件实现的基于 PC 的功能)。进行选择时, 如果所选择的部件存在, WebApp 就会提供。我们会得到每个产品部件的描述信息和价格信息。选择了不同的部件之后, WebApp 会创建并显示一份材料清单。可以给材料清单取一个名字, 并保存起来供将来参考 (见用例保存配置)。

在用例描述中标有下划线的项代表类和内容对象, 它们将被合并为一个或多个 NSU, 使得新客户可以执行选择 SafeHome 部件用例中描述的场景。

图 17-9 描述了用例选择 SafeHome 部件所隐含的导航的部分语义分析。使用前面介绍的术语, 此图也显示了 SafeHomeAssured.com WebApp 的导航方式 (WoN)。重要的问题域类与选中的内容对象 (在此例中, 名为 CompDescription 的内容对象包是 ProductComponent 类的属性) 一同显示, 这些项就是导航节点。每一个箭头代表一个导航链接^①, 并且采用用户触发行为进行标注, 这些行为引发了链接。

WebApp 设计者可以为每个与用户角色相关的用例都创建一个 NSU。例如, SafeHomeAssured.com 的新客户可能有三个不同的用例, 这三个用例都将访问不同的信息及 WebApp 功能, 这就需要为每个用例都创建一个 NSU。

关键点 NSU 描述了每个用例的导航需求。本质上, NSU 表现了角色是如何在内容对象之间或 WebApp 功能之间移动的。

引述 网站导航问题是概念的、技术的、空间的、哲学的及逻辑的问题。因此, 解决办法要求艺术、科学和组织心理学复杂的即兴组合。

Tim Horgan

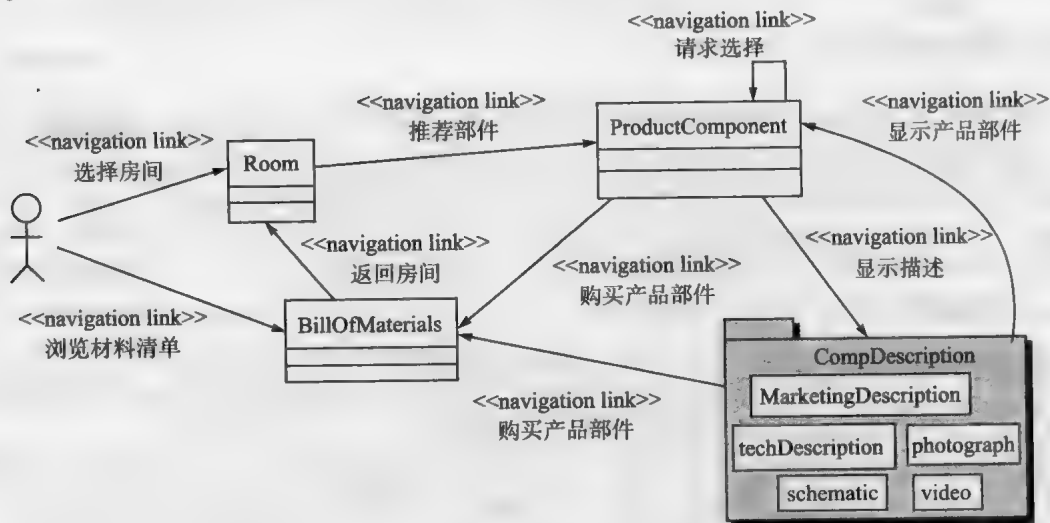


图 17-9 创建 NSU

① 有时将这些导航链接称为导航语义链接 (Navigation Semantic Link, NSL) [Cac02]。

在导航设计的初始阶段，应对 WebApp 的内容体系结构进行评估，为每个用例确定一个或多个 WoN。如上面所说的那样，一个 WoN 标识了导航节点（如内容）和使它们之间能够导航的链接，然后将 WoN 组织到 NSU 中。

17.8.2 导航语法

在进行设计时，下一项任务就是定义导航机制。大多数网站会利用以下一个或多个导航来实现每个 NSU：单独的导航链接，水平或垂直导航条（列表），标签或者一个完整的站点地图入口。

除了选择导航机制以外，设计人员还应该建立合适的导航习惯和帮助。例如，为了使图标和图形链接呈现“可点击”的状态，图标和图形的边缘应成斜角，使其呈现出三维效果。应该考虑设计听觉和视觉反馈，提示用户导航选项已选择。对于基于文本的导航，应该用颜色来显示导航链接，并给出链接已经访问的提示。在进行用户友好的导航设计时，这些仅仅是许多设计习惯中的几种。

建议 在大多数情况下，应选择垂直或水平导航机制，但不要两者都选。

建议 网站地图应该从任何一页都能访问。地图本身应被组织成 WebApp 信息显而易见的结构。

17.9 构件级设计

现代 WebApp 提供了更加成熟的处理功能，这些功能能够：（1）执行本地化的处理，从而动态地产生内容和导航能力；（2）提供了适于 WebApp 业务领域的计算或数据处理能力；（3）提供了高级的数据库查询和访问；（4）建立了与外部协作系统的数据接口。为了实现这些（及许多其他）能力，Web 工程师必须设计和创建程序构件，这些构件在形式上与传统软件构件相同。

第 14 章讨论的设计方法几乎不需要任何修改，就可以适用于 WebApp 构件。实现环境、编程语言、设计模式、框架和软件可能会有些不同，但是，总的设计方法是一样的。

17.10 小结

WebApp 的质量是根据其可用性、功能性、可靠性、效率、可维护性、安全性、可伸缩性及面市时间定义的，它在设计的过程中被引入。为了达到这些质量属性，好的 WebApp 设计应该展现出简单性、一致性、符合性、健壮性、导航性和视觉吸引力。WebApp 设计活动集中在 6 个不同的设计元素上。

界面设计描述了用户界面的结构和组成，包括屏幕布局的表示、交互方式的定义和导航机制的描述。在设计布局 and 界面控制机制时，有一套界面设计原则和界面设计工作流可引导设计者。

美学设计也称平面设计，描述了 WebApp 的“外观和感觉”，包含颜色配置、几何布局、文本字号、字体和位置、图形的使用及相关的美学决策。一套平面设计指导原则为设计方法提供了基础。

内容设计为所有内容定义了布局、结构和外观轮廓，这些内容是 WebApp 的一部分，并建立了内容对象之间的关系。在进行内容设计时，首先对内容对象、它们之间的关联和关系进行描述。一组浏览原语建立了导航设计的基础。

结构设计确定了 WebApp 的总体超媒体结构，并且包含内容结构和 WebApp 结构。内容的结构风格包括线性结构、网格结构、层次结构和网络结构。WebApp 结构描述了使基于

Web 的系统或应用达到其业务目标的基础结构。

导航设计描绘了内容对象之间的导航流和为所有的 WebApp 功能建立的导航流。通过描述一组导航语义单元来定义导航语义。每个单元由导航路径、导航链接和节点组成。导航语法描述用于实现导航的机制，导航也是语义描述的一部分。

388

构件设计开发了实现 WebApp 功能构件所需要的详细处理逻辑。第 14 章所描述的设计技术可应用于 WebApp 的构件工程。

习题与思考题

- 17.1 当构建现代 WebApp 时，为什么仅将“艺术理念”作为设计准则是不够的？是否存在一种情况，在这种情况下艺术理念就是要遵循的准则？
- 17.2 本章我们讨论了 WebApp 的总体质量特性，选择你认为是最重要的三个质量特性，给出论据说明为什么每个质量特性都应该在 Web 工程设计工作中受到重视。
- 17.3 在 17.1 节描述的“WebApp 设计质量检查单”中，至少再增加 5 个问题。
- 17.4 你为一个远程学习公司 FutureLearning Corporation 设计 WebApp。你想实现一个基于 Internet 的能够将课程内容发布给学生的“学习引擎”。此学习引擎为发布任何科目的学习内容（内容设计者将准备合适的内容）提供了基础结构。为学习引擎开发界面设计原型。
- 17.5 你曾访问过的最美观的网站是什么？它为什么美观？
- 17.6 考虑内容对象 Order（订购），一旦 SafeHomeAssured.com 的用户完成了所有部件的选择，并决定购买时，就产生此内容对象。为 Order 及所有合适的设计表示制订 UML 描述。
- 17.7 内容体系结构与 WebApp 体系结构之间的区别是什么？
- 17.8 再考虑习题 17.4 描述的 FutureLearning “学习引擎”，选择一种适合于 WebApp 的内容结构，讨论一下为什么你要这样选择？
- 17.9 在设计习题 17.4 描述的 FutureLearning “学习引擎”时，使用 UML 为所遇到的内容对象开发 3 ~ 4 个设计表示。
- 17.10 对 MVC 体系结构做一些研究，针对习题 17.4 描述的“学习引擎”，看看这种结构是否是合适的 WebApp 体系结构。
- 17.11 导航语法与导航语义之间的区别是什么？
- 17.12 为 SafeHomeAssured.com WebApp 定义 2 ~ 3 个 NSU，对于每个 NSU，给出详细的描述。

扩展阅读与信息资源

Van Duyne 和他的同事（《The Design of Sites》，2nd ed., Prentice Hall, 2007）编写了一本内容全面的书籍，此书覆盖了 Web 工程设计过程的大多数重要方面，并包括设计过程模型及设计模式的详细内容。Johnson（《Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules》，Morgan Kaufman, 2010）已经写了一本关于用户交互设计的书，书中的很多例子都适用于网站设计。Wodtke 和 Gavella（《Information Architecture: Blueprints for the Web》，2nd ed., New Riders Publishing, 2009）、Rosenfeld 和 Morville（《Information Architecture for the World Wide Web》，3rd ed., O'Reilly & Associates, 2006）以及 Reiss（《Practical Information Architecture》，Addison-Wesley, 2000）讲述了内容体系结构和其他主题。

389

虽然“Web 设计”方面的书籍已经有成百上千种，但书中很少讨论适合设计工作的有意义的技术方法。介绍最多的是各种 WebApp 设计的有用的指导原则，展示网页及 Java 程序设计的范例，并讨论对实现现代 WebApp 很重要的技术细节。在这方面的很多书籍中，Butler（《The Strategic

Web Designer 》, How Books, 2013)、Campos (《 Web Design Source Book 》, PromoPress, 2013)、DeFederici(《 The Web Designer's Roadmap 》, Sitepoint, 2012)、Robbins(《 Learning Web Design 》, O'Reilly Media, 2012)、Sklar (《 Principles of Web Design 》, 5th ed., Course Technology, 2011)、Cederholm (《 Bulletproof Web Design 》, 3rd ed., New Riders Press, 2011)、Shelly 和他的同事 (《 Web Design 》, 4th ed., Course Technology, 2011)、Zeldman 和 Marcotte (《 Designing with Web Standards 》, 3rd ed., New Riders Publishing, 2009)、McIntire (《 Visual Design for the Modern Web 》, New Riders Press, 2007)、Watrall 和 Siarto (《 Head First Web Design 》, O'Reilly, 2008)、Niederst (《 Web Design in a Nutshell 》, 3rd ed., O'Reilly, 2006) 以及 Eccher (《 Professional Web Design 》, Course Technology, 2010 和《 Advanced Professional Web Design 》, Charles River Media, 2006) 都很值得参考。

强调美学设计的书有 Beaird (《 The Principles of Beautiful Web Design 》, 2nd ed., SitePoint, 2010)、Clarke 和 Holzschlag (《 Transcending CSS : The Fine Art of Web Design 》, New Riders Press, 2006) 以及 Golbeck (《 Art Theory for Web Design 》, Addison Wesley, 2005) 等, 这些书籍值得在这个主题上经验不多的从业者阅读。

Wallace 和他的同事 (《 Extreme Programming for Web Projects 》, Addison-Wesley, 2003) 介绍了 WebApp 设计 (和其他主题) 的敏捷视图。Conallen (《 Building Web Applications with UML 》, 2nd ed., Addison-Wesley, 2002) 以及 Rosenberg 和 Scott (《 Applying Use-Case Driven Object Modeling with UML 》, Addison-Wesley, 2001) 介绍了使用 UML 对 WebApp 建模的详细例子。

WebApp 设计方面的大量信息可以在网上获得, 最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 的 “software engineering resources” 下找到。

移动 App 设计

要点浏览

概念: 移动 App (MobileApp) 设计包括的技术性活动和非技术性活动有: 建立移动 App 的外观和感觉, 创建用户界面的美学布局, 建立用户交互的规则, 定义总体的体系结构, 开发体系结构中的内容和功能, 并设计移动 App 的内部导航。特别需要注意的是向移动 App 中添加环境感知元素。

人员: 软件工程师、平面设计师、内容开发者、安全专家以及其他利益相关者都参加移动 App 设计模型的创建。

重要性: 设计工作允许你创建模型, 并对该模型进行质量评估, 同时在内容和代码生成、测试及在大多数最终用户参与之前对该模型进行改进。设计是形成移动 App 质量的重要环节。

步骤: 移动 App 设计与 WebApp 设计类似, 包括 6 个主要步骤, 这些步骤由分析建模阶段所获取的信息驱动。对于内

容设计, WebApp 和移动 App 设计都存在同样的问题。在体系结构设计中, 移动 App 开发者需要确定哪些功能将在本地实现, 哪些功能实现为 Web 服务或云服务。界面设计建立的布局和交互机制定义了用户的使用习惯。确保移动 App 对环境的合理使用, 这会直接影响界面设计和内容设计。导航设计定义了最终用户是怎样通过内容结构进行导航的。构件设计描绘了移动 App 功能元素的详细内部结构。

工作产品: 设计模型包括内容、美学、体系结构、界面、导航及构件级设计问题, 它是移动 App 设计的主要工作产品。

质量保证措施: 要对设计模型的每个元素进行评估, 尽力发现错误、不一致或遗漏。另外, 应考虑若干个可选的解决方案, 并对当前设计模型有效实现的程度进行评估。

移动设备, 包括智能手机、平板电脑、可穿戴设备以及其他专业化的产品, 已经掀起了下一波计算浪潮。2012 年 8 月, 洛杉矶时报 [Rod12] 报道称:

一份来自 Chetan Sharma 咨询公司^①的调查报告显示: 有史以来, 美国人使用智能手机的数量首次超过了使用固定电话和功能电话的数量, 同时也显示了美国智能手机市场占有率第一次超过了 50% 的大关。

GigaOM^②指出, 不管是美国排行前四的公司还是那些区域性公司或是小公司, 挖掘市场中更便宜智能手机的潜在价值, 这已经是一种无法逆转的趋势。

关键概念

挑战
云计算
环境感知应用系统
设计
最佳实践
目标
错误
开发移动 App
移动环境

① 详见 <http://www.chetansharma.com/USmarketupdateQ22012.htm>。

② 详见 <http://gigaom.com/mobile/carrier-data-confirms-it-half-of-us-now-owns-a-smartphone/>。

关键概念

质量检查单

技术因素

用户界面设计

GartnerGroup[Gar12] 的报道中指出：在同一季度中，有 4.19 亿部智能手机销往全球，并预计平板电脑的年销售量达到 1.19 亿部，几乎比去年翻了一番。移动计算已经成为主流。

18.1 挑战

当不同的移动设备都有相同的产品功能时，用户就会对自己的移动产品所附带的功能有不同的看法，一些用户希望自己的计算机能有移动设备的功能，另一些用户则关注移动设备给他们带来的自由，而乐于接受移动设备上类似软件产品功能的局限性。还有一些用户依旧期待在传统计算机上或一些娱乐设备上实现不可能的独特体验。良好的用户体验可能比任何移动 App 本身所含有的技术质量方面更为重要。

18.1.1 开发因素

与所有的计算机设备一样，移动平台也因其所交付软件的不同而不同——操作系统（例如安卓或苹果系统）与能够提供广泛功能的成百上千的移动 App 功能的一小部分相结合。现在新的工具允许那些几乎没有经过正规培训的个人去开发或者销售应用产品，与大型软件开发团队开发出来的其他应用一样。

尽管业余爱好者可以开发出应用，但是很多软件工程师认为在目前构造的软件中，移动 App 属于最具挑战性的软件之列 [Voa12]。移动平台很复杂，安卓操作系统和苹果操作系统的代码都超过 1200 万行。移动设备通常都有迷你浏览器，这种浏览器不能展示网页上的全部内容。不同的移动设备通常根据开发环境选择不同的操作系统和平台。移动设备的未来趋势就是比个人计算机更小巧，屏幕尺寸更加多元化。这就需要将更多的注意力放在用户界面设计的问题上，包括如何限制某些内容的显示。除此之外，移动 App 的设计必须考虑到间歇性连接中断、电池寿命的限制以及其他设备约束^①[Whi08]。

提问 什么使移动 App 研发变得不同和具有挑战性？

392

移动 App 运行时，移动计算环境中的系统组件可能会改变其自身的位置。为了保持游动网络^②的连接性，就必须开发用于发现设备、交换信息、维护安全性和通信完整性以及同步动作的协调机制。

此外，软件工程师还必须权衡移动 App 的表现力和利益相关者的安全性问题，以此来确认一个适合的设计方案。为了尽可能节约电池电量，开发者必须努力发现新的算法（或者调整现有的算法）以达到高效节能。可以创建中间件来实现不同型号的移动设备在同一个移动网络里的互相通信 [Gru00]。

关键点 在移动 App 设计中，安全保密性和其他元素之间总有一种折中。

软件工程师应该充分利用设备的特性和情景感知的应用来精心实现用户体验。非功能性需求（例如：安全保密性，性能，可用性）有点不同于网络应用或者桌面应用。用户期望它们能够在大量不同的物理环境下运行，因此这对于移动 App 的测试（第 26 章）增加了一些挑战。因为移动 App 经常运行在各种设备平台上，因此便携性是要着重考虑的。此外，兼容多个平台所耗费的时间和精力往往也会增加整个项目的成本 [Was10]。

① 可在 <http://www.devx.com/SpecialReports/Article/37693> 找到。

② 游动网络与移动设备或服务器保持着持续变化的连接。

18.1.2 技术因素

通过给手机、数码相机、电视等日常设备增加网络功能所耗费的低成本正在改变着人们获取信息和使用网络服务的途径 [Sch11]。在所有的技术因素中, 移动 App 应该采用如下几种处理方式。

多元化的硬件和软件平台。移动 App 运行在具有大量不同层次功能的众多不同的平台上(包括移动平台和固定平台), 这是非常寻常的事情。这些差异存在的部分原因是设备之间的可用硬件和软件是各不相同的, 这增加了开发的成本和时间, 同样也使得配置管理(第 29 章)变得更加困难。

提问 在建立移动 App 时, 最初的技术因素是什么?

多种开发框架和程序设计语言。当前的移动 App 是在至少 5 种流行的开发框架(安卓、苹果、黑莓、Windows、塞班)的基础上使用至少 3 种不同的程序设计语言(Java、Object C、C#)来进行编写的 [Was10]。几乎很少有移动设备允许在设备上直接开发。相反, 移动 App 开发者使用在桌面开发系统上运行的模拟器。这些模拟器有可能精确地反映出设备自身的局限性, 但也许不能。瘦客户端应用往往比专门设计运行在移动设备上的应用更容易移植到多个设备上。

[393]

多种具有不同规则和工具的应用存储库。每一个移动平台都有自己的应用存储库和接入应用标准(例如, 苹果^①、谷歌^②、移动研究公司^③、微软^④和诺基亚^⑤都发布了他们自己的标准)。拥有多个平台的移动 App 的研发必须是分开进行的, 并且每个版本都需要有自己的标准。

极短的开发周期。软件工程师在建立移动 App 时通常采用敏捷开发过程, 以此来尽力减少开发周期 [Was10]。

用户界面的限制以及传感器与照相机之间交互的复杂性。与个人计算机相比, 移动设备拥有更小尺寸的屏幕、更丰富的交互可能性(例如语音、触摸、手势、视线跟踪)和基于环境感知的使用场景。用户界面的风格和外观通常是由特定平台的开发工具的性质而决定的 [Rot02]。允许智能设备与智能空间进行交互, 这一行为提供了创建个性化、网络化、高逼真应用平台的发展潜力, 这些可被理解为将智能手机与车载娱乐信息系统进行合并^⑥。

有效地利用环境。用户期望移动 App 能够基于设备的物理位置及其可利用的网络功能提供个性化的用户体验。用户界面设计和相关环境感知应用将在 18.2 节进行更加详细的讨论。

电源管理。电池寿命通常是移动 App 最重要的限制约束之一。背光、存储器读写、无线网络连接的使用、专业硬件设备的利用以及处理器速度都会影响到电池的使用, 这些都是软件开发者所要考虑的因素 [Mei09]。

安全保密性、隐私模式和方案。无线网络通信很难不被人窃听。事实上, 阻止针对自动应用的中间人攻击^⑦对于用户的安全性来说是至关重要的 [Bos11]。如果移动设备丢失或者

[394]

① <https://developer.apple.com/appstore/guidelines.html>。

② <http://developer.android.com/distribute/googleplay/publish/preparing.html>。

③ <https://appworld.blackberry.com/isvportal/guidelines.do>。

④ <http://msdn.microsoft.com/en-us/library/ff941089%28v=vs.92%29.aspx>。

⑤ <http://support.publish.nokia.com/?p=64>。

⑥ 在汽车设置中应用智能设备时, 应该能够限制可能会分散驾驶员注意力的服务接入, 并在车辆移动过程中允许免提操作 [Bos11]。

⑦ 这些攻击涉及第三方对两个可信来源之间的通信拦截, 并冒充一方或双方当事人进行通信。

被人下载了恶意应用程序，那么存储在设备上的数据就会被盗窃。那些用来提高移动 App 安全保密及隐私方面可信度的软件方案通常降低了应用的可用性和用户之间相互交流的自发性 [Rot02]。

计算和存储限制。使用移动设备来控制家庭环境和安全保密服务是人们关注的一个领域。允许移动 App 在它们的环境中与设备和服务进行交互时，移动设备将会很容易地被海量的信息占据（例如存储、处理速度、能量消耗）[Spa11]。开发人员可能需要寻找减少处理器与内存资源占用的编程技巧和方法。

依赖外部服务的应用。瘦移动客户端的建立表明，需要依靠网络服务提供商和云存储设施，这增加了人们对数据或服务的可访问性与安全性的担忧 [Rot02]。

测试的复杂性。瘦客户端移动 App 对于测试来说是一个特殊的挑战^①。虽然对于 WebApp 的测试存在很多挑战（第 25 章），但人们却把更多的关注点放在利用互联网网关和电话网络的数据传输上 [Was10]。移动 App 的测试在第 26 章进行详细讨论。

18.2 开发移动 App

Andreu[And05] 描述了一个移动 App 设计的螺旋过程，该过程模型包含 6 个步骤。

构想。确定移动 App 体系结构设计、导航设计、目标、特征和功能，以此来确定第一增量的范围和规模。开发人员必须意识到人类、社会、文化及组织的活动。这些活动有可能揭示用户的潜在要求并影响移动 App 的商业目标和功能。

关键点 WebApp 开发利用敏捷、螺旋工程过程模型。

计划。确认整个项目的成本和风险。制定详细的进度表，并编写下一个增量的处理过程。

分析。详细说明所有移动用户的需求，确认所需的内容条目，包括内容分析、交互分析、功能分析和配置分析。在这个阶段，开发人员要确认创建瘦客户端还是富客户端。识别用户目标（信息方面、业务方面）的特征将会有助于确认所要开发的移动 App 的类型。

[395]

工程。涉及体系结构设计、导航设计、界面设计、内容设计和内容制作。软件工程师检查目标移动设备的约束条件，包括选择的无线网络技术以及实现移动 App 所需要的 Web 服务的特征。

实现和测试。在这个步骤中，要对移动 App 进行编码和测试。所有问题中，使测试具有挑战性的有以下几项：（1）由无线电干扰而导致的高损失率和因网络覆盖问题而引发的频繁的连接中断；（2）由相对较低的带宽造成的经常性数据传输延迟；（3）因为移动设备安全性低，相对容易遭到攻击而产生的安全问题。

用户评估。评估移动 App 的可用性和可访问性，然后制定下一个增量的过程计划。

Andreou[And05] 建议，普遍性、个性化、灵活性以及本地化应该是每一个移动 App 的主要设计目标。移动用户希望拥有实时接收信息及实时处理事务的能力，而忽略物理位置或者并发用户数。移动 App 应该呈现的是根据用户喜好定制的服务和应用程序。移动用户应该能够轻松地完成获取信息或事务处理等操作。移动用户应该能够获取本地信息和服务。这意味着在设计移动 App 用户体验时利用相关环境的重要性。

提问 在评估移动 App 的质量时我们应该考虑什么？

① 在设备上完整运行的移动 App 可以使用传统的软件测试方法（第 23 章），或使用个人计算机上运行的模拟器进行测试。

SafeHome 制定移动 App 需求

[场景] 会议室，确定 SafeHome WebApp 移动版本需求的第一次会议。

[人物] Jamie Lazar，软件团队成员；Vinod Raman，软件团队成员；Ed Robbins，软件团队成员；Doug Miller，软件工程项目经理；三名市场营销部成员；一名产品工程代表；一名主持人。

[对话]

主持人（指着白色写字板）：这就是当前存在于 WebApp 中的住宅安全功能的对象和服务列表。

Vinod（打断）：我的理解是人们希望通过移动设备访问 SafeHome 功能……包括住宅安全功能？

市场营销部负责人：是的，就是这样……我们必须增加这个功能，并尝试使其自适应环境，帮助实现个性化的用户体验。

396

主持人：什么场景下的自适应环境？

市场营销部负责人：当人们在开车回家的路上时，他们可能希望使用智能手机，而不是控制面板，要避免登录网站。或者他们可能不希望所有的家庭成员都可以访问其手机系统的主控制面板。

主持人：你考虑了特殊的移动设备吗？

市场营销部负责人：嗯，所有的智能手机都可以。我们有一个 Web 版本的移动 App 将要完成了，为什么不在所有的智能手机上面运行呢？

Jamie：不太可行。如果我们采用移动手机浏览器的方法，我们可能会重用许多的 WebApp。但是请注意，智能手机的屏幕尺寸变化多样，并且它们可能不都具有相

同的触摸功能。因此，考虑到设备特征，至少我们得建立一个移动网站。

Ed：也许我们首先应该建立网站的移动版本。

市场营销部负责人：可以，但移动网站方案不是我们的初衷。

Vinod：似乎每个移动平台也都有自己的独特开发环境。

产品代表：我们可以将移动 App 开发限制在一种或两种类型的智能手机上吗？

市场营销部负责人：我认为可行。现在的智能手机市场是以两种智能手机平台为主。

Jamie：还要考虑安全问题。我们要确保外来人员无法入侵系统，攻破、劫持或者发生更糟的事情。而且手机也可能比笔记本电脑更容易丢失或被盗。

Doug：说得对。

市场营销部负责人：但是，我们仍然需要相同的安全级别……确保可以阻止外来人员利用偷来的手机进入。

Ed：说起来容易，做起来难……

主持人（打断）：我们不要担心这些细节。

（Doug，作为会议的记录人，做了相应的注释）

主持人：以此为起点，我们能否确认在移动 App 中需要哪些 WebApp 安全功能元素，哪些需要重新开发？这样我们就能够决定可以支持多少移动平台，进而向前推进这个项目。

（小组接下来花费了 20 分钟提炼和扩展住宅安全功能的详细内容。）

18.2.1 移动 App 质量

实际上，在第 19 章中讨论的几乎每个质量维度和因素都可应用于移动 App。然而，Andreou[And05]指出，最终用户对于移动 App 的满意度取决于 6 个重要的质量因素：功能性、可靠性、易用性、效率、可维护性及可移植性。

信息栏 移动 App——质量检查单

下面的检查单提供了一组问题，可以帮助软件工程师和最终用户评估整体移动 App 的质量。

- 可以按照用户的喜好定制内容、功能或导航选项吗？
- 可以根据用户通信的带宽定制内容或功能吗？应用对信号较弱或信号丢失的处理是否可以接受？
- 可以按照用户的喜好使内容、功能或导航选项做到自适应环境吗？
- 充分考虑过目标设备上的电池容量吗？
- 合理利用了图形、媒体（音频、视频）

和其他 Web 服务或云服务吗？

- 整个页面设计是否易于阅读和浏览？应用程序是否考虑了屏幕尺寸的差异？
- 用户界面符合针对目标移动设备的显示和交互标准吗？
- 应用符合用户期望的可靠性、安全性和私密性吗？
- 采取了什么措施以确保应用能够保持发展趋势？
- 移动 App 已经在所有的目标用户环境和所有的目标设备上进行测试了吗？

397

18.2.2 用户界面设计

移动设备用户希望能够用最短的时间来学习并掌握一个移动 App。为了达到这个目标，移动 App 设计者应在不同的平台上使用统一的图标来展示和布局。此外，设计者必须十分敏感，更加关注用户对隐私方面的期望，这些属于隐私的个人信息会显示在移动设备屏幕上。触摸和手势界面以及先进的语音输入正在迅速走向成熟 [Shu12]，并已成为用户界面设计师工具箱的一部分。

为所有人提供访问权限所产生的法律和道德压力表明：移动设备界面需要考虑品牌差异、文化差异、计算体验差异以及老年用户和残疾人用户（例如，视觉障碍、听觉障碍、行动不便者）。可用性差就意味着用户不能完成他们的任务或者不满意结果。这也表明，在每个可用性区域中（用户界面、外部辅助界面和服务界面），以用户为中心的设计活动的重要性。为了满足利益相关者对于可用性的期望，移动 App 开发者应该尝试回答这些问题以评估设备的外部表现：

- 用户界面在多个应用中是否一致？
- 设备是否能与不同的网络服务相互协作？
- 在目标市场中，就利益相关者的价值观[⊖]而言，设备是否能被大众所认可？

Eisenstein[Eis01] 声称运用抽象的、与平台无关的模型来描述用户界面极大地促进了移动设备多平台用户界面一致性和可用性的发展。所谓的基于模型的设计使用了三种不同的模型。平台模型描述了所支持平台的约束条件。表示模型描述了用户界面的外观。任务模型就是用户需要平台完成任务目标的一种结构化表示。在最好的情况下，基于模型的设计（第 12

关键点 应用以用户为中心的设计时，可访问性是一个重要的设计问题，必须考虑。

引述 移动 App 迫使设计师深入了解用户需求，以便提供可以学习和使用界面的正确功能。

Ben
Schneiderman

⊖ 品牌，伦理偏好，道德喜好，认知信仰。

章) 涉及包含建模的数据库创建, 并具有为多个设备自动生成用户界面的工具支持。利用基于模型的设计技术还可以帮助设计人员识别并适应独特的环境以及当前移动计算中的环境改变。没有用户界面的抽象描述, 移动用户界面的开发可能容易出错并耗费更多的时间。

398

信息栏 移动 App 用户界面设计注意事项

设计选择会影响性能, 应在用户界面设计过程的早期进行检查。Ivo Weevers [Wee11] 给出了一些在设计移动 App 时有益的方法。

- **定义用户界面的品牌标志。**使自己的应用区别于竞争对手的应用。使品牌的核心标志性元素具有最好的响应, 因为用户会反复使用它们。
- **注重产品的结合。**确定平台目标。平台对于应用和公司的成功是最重要的, 并非所有平台都具有相同的用户数。
- **确定核心用户案例。**利用相关技术(例如质量功能部署(第8章))来减少冗余的需求列表, 以使用移动设备上可用的受限资源。
- **优化用户界面流程和元素。**用户不喜欢等待。找出用户的工作流程中潜在的瓶

颈, 出现延迟时确保给出用户进度指示。就用户的利益而言, 确保屏幕元素的显示时间是合理的。

- **定义缩放规则。**当要显示的信息太多, 屏幕无法容纳时, 确定将要使用的选项。管理功能、美观性、可用性和性能, 使其保持持续的平衡。
- **使用性能仪表盘。**经常就产品的当前完成状态进行沟通(例如, 已完成的用户案例的数量), 还要沟通当前产品相对于目标的性能参数, 也许还会与竞争对手进行比较。
- **用户界面设计技巧的关键。**理解布局、图形和动画的使用对性能产生的影响是很重要的。元素显示的渲染和程序执行的交错结合手法可能会很有帮助。

18.2.3 环境感知 App

环境允许基于移动设备的位置和移动设备具有的功能性来创建新的应用软件。它还能帮助调整个人计算机的应用来适应移动设备(例如, 当家庭卫生保健工作人员到达患者房间时, 他所携带的移动设备能自动下载患者的信息)。

采用适应性高、与环境相关的接口是一种很好的处理设备局限性的方法(例如, 屏幕尺寸和内存)。为了更好地开发环境感知的用户交互, 需要相应的软件体系结构的支持。

在早期的环境感知应用的讨论中, Rodden[Rod98] 指出: 移动计算通过提供允许设备感知自身位置、时间和周围物体的功能, 将现实世界和虚拟世界连接在一起。该设备可以在一个固定的位置, 如警报传感器一样, 嵌入到独立的设备中, 或者由人随身携带。因为设备可用于个人、团体或大众, 所以它必须监测并识别用户的存在和身份, 以及用户所依赖或准许的相关环境属性(即使这个用户是另外一台设备)。

为了实现环境感知, 移动系统一定要从各种不确定、快速变动的异构数据源中生成可靠的信息。由于噪音、误差、磨损和气候的原因, 通过梳理多个传感器的数据来提取相关的环境信息是具有挑战性的。在相关环境感知系统中, 基于事件的通信对高度抽象、连续性数据流的管理来说是非

提问 设计是如何获得环境感知的?

399

常适合的 [Kor03]。

在普适计算环境中，大部分用户使用大量不同的设备工作。由于移动工作实践的需求，设备的配置要足够灵活以便能经常性地地进行改变。这对软件基础设施实现对不同类型交互的支持是很重要的（例如，手势、声音和笔），并能抽象存储，容易共享。

有些时候，用户可能期望使用多个设备同时作用于一个产品（例如，使用触屏设备来编辑文档的图像，同时使用键盘来编辑文档的文本）。整合众多不总是连接到网络并具有各种限制的移动设备是很具有挑战性的 [Tan01]。需要连网的多人游戏都不得不面对这样的问题，它们需要在设备上存储游戏状态并共享其他游戏玩家设备上实时更新的信息。

18.2.4 经验教训

本章前面我们提到了一些移动App和传统软件之间的重要区别。由于存在这些差异，软件工程师需要修改和扩展传统技术，以便分析、设计、构建和测试移动App。de Sá 和 Carrico[Des08]提出了一些经验教训。

移动App的使用场景（第15章）必须考虑相关环境因素（位置、用户和设备）以及相关场景之间的转换（例如，用户从卧室移动到厨房或使用手指代替触控笔）。de Sá 和 Carrico 已经确定了一组用户场景开发中应该考虑的变量类型——位置和设置、运动和姿势、设备和用法、负载和干扰以及用户的喜好。

建议 在开发移动App的过程中，用例可以发挥很好的作用，但是开发时必须考虑相关环境因素。

人种观察^①是一种广泛使用的方法，用来收集所设计的软件产品的具有代表性的用户信息。随着用户场景的改变，观察他们通常是很困难的，因为观察者必须跟随用户很长一段时间，而这可能引发隐私问题^②。一个复杂的因素就是，有时用户在私人场合和公开场合完成的任务是不同的。随着场景的变化，可能需要观察同一用户完成不同场景下的任务，同时记录用户对于变化的反应。

早期的移动App用户界面原型可以在图纸上创建，使用草图或者索引卡和贴字条的组合来模拟重要的互动机制。关键是允许评估所有的交互机制，检查所有的使用环境，并详细说明所有的用户交互机制。除此之外，还要模拟互动小工具的使用以及整体的屏幕布局 and 定位。在目标移动设备发挥作用之前，这些早期的粗糙图纸原型可以协助发现错误、不一致和遗漏。一旦解决了布局和定位问题，就可以创建后面的原型，并在目标移动设备上运行。

400

信息栏 移动App设计误区

Joh Koester[Koe12]列出了移动App设计实践中应该避免的一些情况。

- **功能复杂。**避免在应用中添加太多的功能、在屏幕上添加太多的部件。简单是可以理解的。简单是适合市场的。
- **前后矛盾。**为了避免这种情况，应在设

置页面导航、菜单使用、按钮、选项卡和其他用户界面元素的标准时保持统一的外观和风格。

- **设计过度。**设计移动App时要有狠心。删除不必要的元素和多余的图形。不要试图添加仅仅是因为你觉得应该添加的

① 人种观察是一种通过观察工作环境中的用户来确定用户任务性质的手段。

② 在直接观察不能实现时，邀请用户填写匿名问卷也是可以的。

元素。

- **加载过慢。**用户不关心设备的限制因素，他们只希望快速查看。尽可能进行预加载。去除不需要的东西。
- **废话连篇。**冗长的文字菜单和屏幕展示表明这是一个没有经过用户测试的移动 App，开发者也没有花费足够的时间理解用户的任务。
- **非标准的交互。**以平台为目标的原因之

一就是利用用户在平台上完成事情的体验方式。标准存在于使用过程之中。这需要在可能的情况下，满足应用在不同设备上展示相同的外观和操作行为的需求，以达到平衡。

- **帮助和常见问题回答。**加入联机帮助不是修复设计不佳的用户界面的方式。确认你已经和目标用户一起测试了应用，并修复了所识别的缺陷。

18.3 移动 App 设计的最佳实践

对于移动 App^①的研发，以及像苹果的 iOS^②或谷歌的 Android 等特殊平台上的应用研发，它们都是有若干准则的^③。Schumacher[Sch09] 已经收集了许多最佳实践的想法并发布了一些特别适用于移动 App 设计的 Web 页面^④。在设计移动触屏应用软件时，Schumacher 列出了如下一些重要的因素。

- **确定受众。**应用开发要融入用户心目中的期望和背景。有经验的用户希望快速地做事。缺乏经验的用户在第一次使用应用时会期望有手把手的熟悉过程。
- **根据使用环境进行设计。**在使用移动 App 时，考虑用户将如何与现实世界互动是非常重要的。和你离开办公室前检查天气相比，在飞机上观看电影需要不同的用户界面。
- **在简单和懒惰之间有一条很好的界线。**相比于简单地移除在较大设备上运行的应用程序用户界面上的功能，在移动设备上创建直观的用户界面更加困难。用户界面应该提供使用户做出下一个决定的所有信息。
- **将平台作为优势。**触摸屏导航不直观，所有新用户必须学习。如果用户界面设计者坚持使用已有的平台标准，那么学习任务将会比较容易。
- **使醒目的滚动条和选择项更加突出。**可触摸设备上的滚动条很难定位，因为它们太小。确保菜单或图标的边界足够宽，使得颜色变化可以吸引用户的注意力。在使用颜色编码时，确保前景色和背景色之间有足够的对比度，让色盲用户也都能够区分它们。
- **提高高级功能被用户发现的概率。**包含在移动 App 中的热键和其他快捷键可以使有经验的用户更快速地完成任任务。可以通过包含在用户界面的视觉设计线索来增加这些功能被用户发现的可能性。
- **使用明确和一致的标签。**不考虑特定平台所使用的标准，小部件的标签应该被所有

提问 在设计移动触屏 App 时应该考虑什么？

① <http://www.w3.org/TR/mwabp/>。

② <https://developer.apple.com/library/iOS/navigation/>。

③ <http://developer.android.com/guide/components/index.html>。

④ <http://www.usercentric.com/news/2011/06/15/best-practices-designing-mobile-touch-screenapplications>。

应用系统的用户识别。谨慎地使用缩写，尽量避免使用。

- 巧妙的图标不应以牺牲用户的理解而设计。图标有时仅仅对于他们的设计师来说是有意义的。用户必须能够快速了解它们的含义。很难保证图标在所有的语言和用户组中都有意义。提高识别度的一个好策略是在一个新的图标下面添加一个文本标签。
- 支持用户期望的个性化。移动设备用户希望能够个性化设置一切。至少，开发者应该尽力允许用户设置其位置（或自动地检测到），并选择在该位置可利用的内容选项。提示用户哪些功能可以进行个性化设置以及如何进行设置是非常重要的。
- 在移动设备上长滚动条形式优于多个屏幕的显示。有经验的移动设备用户希望所有信息都展示在一个单一的输入画面上，即使利用滚动条。新手往往会迅速变得经验丰富并厌弃多屏幕输入。

402

开发多个设备平台的本地应用可能是昂贵和费时的。在移动设备上通过使用 Web 开发人员熟悉的技术（例如，JavaScript、CSS 和 HTML）创建使用 Web 浏览器进行访问的移动 App 可以减小开发成本。开放式 webOS^①就是一个旨在允许这种类型开发的独立于设备的平台。

18.4 移动开发环境

开发环境工具栏上的工具可用于针对主流平台开发移动 App。每个工具都有自己的优点和缺点^②。有的技术受限于单一厂商设备（例如，iOS 和 Objective C），有的平台授权给几家制造商（例如，Android 和 Java，Windows 8 和 C#），有些是开源的，可运行在许多设备上（例如 webOS 和 Enyo）。每个平台都有自己的营销和分布规则，每一级别的变化都支持特定的应用技术，就像游戏一样。

信息栏 移动 App 开发工具

许多为流行设备开发移动 App 的工具都可以从网上免费下载^③：

- <https://developer.apple.com/resources>。苹果的 iOS 开发中心包含可用于开发 iPod、iPad 和 iPhone App 的工具。
- <http://developer.android.com/index.html>。该网站提供了一个允许 Android 开发使用 Eclipse 编程环境的插件。
- <https://developer.blackberry.com/java/download/eclipse/>。该网站提供了一个允许黑莓开发使用 Eclipse 编程环境的插件。
- <http://eclipse.org/>。Eclipse 编程环境下载站点。
- <http://create.msdn.com/en-US/>。微软 Windows Phone App 和 Xbox 游戏开发工具。
- http://www.developer.nokia.com/Develop/Java/Tools/Series_40_platform_SDKs/。几个基于 Java 的诺基亚开发工具下载网站。
- <https://developer.palm.com/content/resources/>。HP webOS 开发环境下载网站。
- <http://enyojs.com/>。Enyo 跨平台开发环境下载网站。
- <http://www.scirra.com/construct2>。Construct2 跨平台游戏开发环境下载网站。

① 关于 webOS 的详细信息可以在 <https://developer.palm.com> 找到。

② 在 http://www.cs.colorado.edu/~kena/classes/5828/s10/presentations/software_engineering_mobile.pdf 可以找到更多讨论。

③ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

平台（或多个平台）的选择需要移动 App 开发者认真思考。有时候，选择的平台将会由客户的业务目标来决定。在其他情况下的平台选择则是由他们支持的设备功能和已存在的硬件限制来决定。Yuan[Yua02] 使用下列标准来评估一些移动交互式开发环境（Mobile Interactive Development Enviroment, MIDE）。

- **通用的生产特征。**此 MIDE 应包含支持编辑、项目管理、调试、体系结构设计、文档编写和单元测试的工具。
- **第三方 SDK 集成。**每个网络或云服务都可能需要使用特定的 API（Application Programming Interface）或 SDK。在一个交互式开发环境 IDE（Interactive Development Enviroment）而不是多个 IDE 中持续工作是比较容易的。
- **后编译工具。**一个高效的 MIDE 包含优化工具，用以优化已完成的特定移动设备或服务的应用源代码。
- **自动部署支持。**一个好的 MIDE 应该允许测试在开发环境中部署的应用。当移动 App 需要访问 Web 服务或其他应用时，这可能比较棘手。
- **端到端的移动 App 开发。**移动设备往往没有足够的能力来处理或存储大量的本地信息。允许开发人员使用桌面 MIDE 创建、测试和部署整个移动 App 项目是非常重要的。
- **使用文档和教程。**即使是免费的开发工具也必须容易学习和易于使用。足够的支持材料是必不可少的。
- **图形用户界面构建。**如果 MIDE 支持用户界面的可视化构建，那么原型可以快速构造并测试。

提问 我应该如何去选择移动环境和平台？

正如我们已经指出的那样，设计和实现在多种平台上连续运行的移动 App 是很难的。这对于开发手机游戏（一个价值数十亿美元的产业）应用尤其如此。最流行的游戏都是为多个移动设备并行开发。这种类型的开发带动成本上升，因此需要更好的标准化开发工具和 API。Galavas [Gal11] 指出，在考虑使用哪一个移动开发平台时，可移植性、功能性、开发速度和性能都是关键的选择标准。

移动计算中间件可用于促进分布式系统组件的通信和协调。这可以使移动 App 开发者依赖于这些隐藏了一些移动环境复杂性细节的抽象对象。中间件在移动 App 开发中是很有用的，移动客户端和移动服务提供商均允许异步间歇式连接。在移动客户端上运行的中间件不能消耗移动设备上的重要计算资源。中间件还必须帮助移动 App 实现用户所要求的环境感知级别 [Mas02]。

信息栏 移动 App 中间件

以下中间件是为移动 App 而开发的具有代表性的产品。

- <http://www.infracore.com/products/mobi>。Mobi 移动中间件是一组库和 WSGI 组件，作用于 Web 服务器和提供移动数据的应用之间。
- <http://smartsoftmobile.com/>。SmartSoft

移动解决方案提供了基于云计算和企业（如 SAP）的移动设备平台解决方案。

- <http://www.sybase.com/>。Sybase 提供了移动企业应用平台（MEAP），为移动 App 和企业应用的开发提供工具和客户端服务器中间件。也可以参考 <http://scn.sap.com/community/mobile>。

- <http://code.google.com/p/skeenzone/>。
SkeenZone 是轻量级的、可扩展的 Java 中间件，支持分布式移动 App 的开发。
- <http://modolabs.com/platform>。Kurogo

是一个开源平台，用来丰富内容多样性、多层面的移动网站以及 iPhone 和 Android 应用。

18.5 云

服务计算^①和云计算^②实现了大规模分布式应用的快速开发 [Yau11]。这些计算模式可以更容易、更经济地创建不同设备（个人计算机、智能手机和平板电脑）上的应用。这两种模式允许资源外包或将信息技术管理的信息转移给服务提供商，而同时减轻资源限制在某些移动设备上的影响。面向服务的体系结构提供了移动 App 开发所需要的体系结构风格（例如 REST^③）、标准协议（例如 XML^④、SOAP^⑤）和接口（例如 WSDL^⑥）。云计算可以方便、按需地通过网络访问可配置的计算资源（服务器、存储、应用程序和服务）共享池。

服务计算使移动 App 开发人员避免了将服务源代码集成到运行在移动设备上的客户端的需求。取而代之的是，服务在提供商的服务器上运行，并与通过消息传递协议利用它的应用保持松耦合状态。服务通常会提供 API，这样我们就可以把它当作一个抽象的黑盒子来处理。

云计算响应网络内任意地方、任意时间的客户端（用户或程序）对所需计算能力的请求。云计算的架构有三层，每一层都可以称为一个服务。软件即服务包括由第三方服务提供商提供的软件构件和应用。平台即服务提供了一个协同开发平台，协地理上分散的团队成员进行设计、实施和测试。基础设施即服务为云计算提供虚拟计算资源（存储、处理能力、网络连接）。

移动设备可以从任何位置在任何时间访问云服务。身份盗窃和服务劫持风险的存在使得移动服务和云计算提供商需要采用严格的安全工程技术来保护他们的用户。与云计算相关的安全和隐私问题将在第 27 章中讨论。采用中立厂商的云服务可以更容易地创建跨平台的应用程序 [Rat12]。

Taivalsaari[Tai12] 指出，利用云存储可以让世界各地数以百万计的任何移动设备或软件功能很容易地获得更新。事实上，虚拟化整个移动用户体验，使所有应用软件都从云下载是可能实现的。

引述 面向服务的软件工程集成了服务计算和云计算模式的最佳功能。

Stephan Yau

引述 由于用户的业务严重依赖于第三方服务提供商，所以服务的可靠性威胁如何会影响服务及云用户业务受到了严重关切。

Stephan Yau

① 服务计算专注于体系结构设计，通过服务发现和服务组合来实现应用的开发。

② 云计算专注于通过灵活的、可扩展的资源虚拟化和负载均衡，有效地为用户提供服务。

③ 表述性状态转移（REST）描述了一种网络上的 Web 体系结构风格，其中资源的表现形式（例如 Web 页面）将客户端置于一个新的状态。客户端的状态随资源的表现形式一起改变或转移。

④ 可扩展标记语言（XML）用来存储和传送数据，而 HTML 用来显示数据。

⑤ 简单对象访问协议（SOAP）是在计算机网络中用于实现 Web 服务结构化信息交换的协议规范。

⑥ Web 服务描述语言（WSDL）是一种基于 XML 的语言，用于描述 Web 服务以及如何访问它们。

18.6 传统软件工程的适用性

我们不能保证桌面程序或 WebApp 可以很容易地实现为移动 App。然而,许多用于创建桌面应用的敏捷软件工程实践(第 5 章)可以用来创建独立的移动 App 或移动客户端软件,许多用来创建优质 WebApp 的做法也可以用来创建移动 App 使用的 Web 服务。

在制定目标和用户案例时可以融合许多敏捷过程模型中使用的做法。这将确定所需要的用户体验,确定利益相关者必须满足的需求,确定移动 App 必须考虑的环境变量。在确定桌面或 WebApp 目标时,不太可能考虑到环境和位置感知的作用。

在规划活动时,为一个以上的设备或平台进行开发所遇到的困难必须被计入项目预算和时间表,以便所有利益相关者所担忧的必需资源得到适当分配。开展有意义的可用性测试和充分的现场测试的困难度增加了移动 App 的开发成本。如果发生安全事件或隐私侵犯,风险分析应该考虑损失的影响。制定需求安全评估计划是明智的(第 27 章)。

在移动 App 开发中,上线时间是至关重要的。除此之外,新的科技元素和不断变更的用户需求往往随着开发资金一起引入。正如我们前面提到的,迭代敏捷过程模型(第 4、5 章)为开发者提供机会,可以调整并适应那些基于不断进化的产品原型进行评估的需求。

在产品设计中,内容分析和设计与构建 WebApp(第 17 章)时,所应用的操作类似。包含在移动 App 里的内容受限于目标设备和平台,需要进行选择和割舍。

通过使用迅速扩张的集合设计模式(第 13 章)和面向移动 App [Mes08] 的基于组件的设计(第 14 章),移动 App 的设计可以更加快速。当现有的服务并入移动 App 时,就会采用基于构件和面向对象设计的组合策略。不影响服务质量的复用是移动 App 开发的一个核心目标 [Zha05]。

用户界面设计大量借鉴了 Web 页面的图形和美学设计经验教训(第 17 章),用以支持移动 App 的品牌目标。以用户为中心的设计,其重点是易用性和可访问性,用户界面对于创建优质的移动 App 是非常重要的(第 15 章)。

最重要的体系结构设计决策往往是能否建立一个瘦客户端或富客户端。模型-视图-控制器(MVC)架构(第 17 章)普遍应用于移动 App 中。因为移动 App 架构可能对导航有很大的影响,所以在设计行为期间做出的决定将会影响导航设计期间工作的进展。体系结构设计必须考虑到设备资源(存储、处理器速度和网络连接)。设计应包括提供可发现服务和可移动设备的相关内容。

可用性测试和部署测试存在于每个原型开发周期。着眼于安全问题的代码审查应作为执行活动的一部分。这些代码审查应根据系统设计活动来确定相应的安全目标和威胁。安全性测试是系统测试的常规部分(第 22 章)。

18.7 小结

就功能性、可靠性、可用性、效率、安全性、可维护性、可扩展性而言,移动 App 质量的定义是在设计过程中引入的。一个好的移动 App 应基于以下设计目标:简单、普遍性、个性化、灵活性和本地化。

界面设计描述了用户界面的结构和组织,包括屏幕布局的表现、互动模式的定义以及导航机制的描述。另外,一个良好的移动 App 界面将提升品牌的影响力并聚焦在它的目标设备平台上。一套核心用户案例用来修剪应用中不必要的特性,以管理其资源需求。环境感知

设备利用发现服务来帮助定制个性化的用户体验。

内容设计是非常重要的，并要考虑到屏幕和移动设备的其他限制因素。美学设计也叫平面设计，描述了移动 App 的“外观和感觉”，包括配色方案、平面布置、图形的使用及相关的美学判定。美学设计还必须考虑设备的局限性。

体系结构设计标识出移动 App 的超媒体结构全貌，包括内容架构和移动 App 架构。确定移动 App 中有多少功能留在移动设备上、有多少功能是由网络或云服务来提供是至关重要的。

导航设计代表着内容对象和所有移动 App 功能之间的导航流程。导航语法由目标移动装置上的可用部件定义，语义通常由移动平台确定。内容分块必须考虑间歇性的服务中断以及用户需求的快速性能。

408

构件设计开发实现了用于构建一个完整的移动 App 功能部件所需的详细处理逻辑。第 14 章中描述的设计技术也许可以应用于移动 App 构件工程。

习题与思考题

- 18.1 解释为什么决定为多个设备开发移动 App 可能是一个昂贵的设计决策。有没有一种办法可以减轻支持错误平台的风险？
- 18.2 在本章中我们列出了许多移动 App 的质量特性，选择你认为最重要的三个质量特性，说明为什么这三个特性应该在移动 App 设计工作中受到重视。
- 18.3 在 18.2 节描述的“移动 App 设计质量检查单”中，至少再增加 5 个问题。
- 18.4 你是项目规划公司（Project Planning Corporation，一个开发生产力软件的公司）的移动 App 设计师。你想要实现相当于一个数字三环活页夹的应用系统，可以让平板电脑用户组织和整理几种用户自定义标签下的电子文档。例如，厨房改造项目可能需要一个 PDF 目录、一个 JPG 或 DFX 设计图纸、一个 MS Word 方案和橱柜选项卡下保存的 Excel 电子表格。一旦设计完成后，活页夹及其标签页的内容可以存储在平板电脑或一些云存储上。此应用需要提供 5 个关键功能：活页夹和标签页的定义；从网络位置或设备获取电子文档；活页夹管理功能；标签页显示功能；允许便利贴添加到任意页面的笔记功能。为这个三环应用开发设计图形界面并作为图纸原型进行实施。
- 18.5 你曾使用过的最美观的移动 App 是什么？它为什么美观？
- 18.6 为习题 18.4 描述的三环应用创建用户案例。
- 18.7 使三环应用成为环境感知的移动 App 可能要考虑什么？
- 18.8 反思习题 18.4 描述的 ProjectPlanning 三环应用，为第一个工作原型选择开发平台。讨论你为什么做出这样的选择。
- 18.9 在设计习题 18.4 描述的三环应用时，使用 UML 来开发设计所遇到界面对象的表现形式。
- 18.10 做一些额外的 MVC 架构研究，对于 18.4 题描述的三环应用，论证其是否是一个合适的移动 App 体系结构。
- 18.11 描述三个有希望添加到 SafeHome 移动 App 的环境感知功能。
- 18.12 做一些互联网调查，找出一个支持移动 App 的中间件产品。描述这个中间件的功能和它支持的平台。

扩展阅读与信息资源

Kumar 和 Xie 编写的书（《Handbook of Mobile Systems Applications and Services》，Auerbach Publications，2012）涵盖了移动服务和移动计算中面向服务架构的作用。普适计算方面的书包括 Adelstein（《Fundamentals of Mobile and Pervasive Computing》，McGraw-Hill，2004）和 Hansmann

409

(《Pervasive Computing: The Mobile World》, 2nd ed., Springer, 2003, 2013 年重印), 书中定义了移动计算方面的原则。Kuniavsky (《Smart Things: Ubiquitous Computing User Experience Design》, Morgan Kaufman, 2010) 和 Polstad (《Ubiquitous Computing: Smart Devices, , Environments and Interactions》, Wiley, 2009) 描述了智能设备、智能环境和智能交互之间的环境感知计算。Neil (《Mobile Design Pattern Gallery》, O'Reilly, 2012) 用来自 6 个移动平台的例子来说明移动设计模式。

关于界面设计的书有很多。Schumache (《Handbook of Global User Research》, Morgan-Kaufmann, 2009) 编写了一本比较好的通用参考书。Hoover (《Designing Mobile Interfaces》, O'Reilly, 2011) 描述了一种用于构成页面、显示信息和利用传感器的与设备无关的最佳实践。Nielsen (《Mobile Usability》, New Riders, 2012) 提供了关于如何考虑移动设备的屏幕尺寸以设计可用界面的建议。Colborne (《Simple and Usable Web, Mobile, and Interaction Design》, New Riders, 2010) 描述了简化用户交互的过程。Ginsburg (《Designing the iPhone User Experience: A User-Centered Approach to Sketching and Prototyping iPhone Apps》, Addison-Wesley, 2010) 讨论了从竞争中区分你的应用的重要性, 这是通过采取以用户为中心的方法设计高品质的用户体验。

《Microsoft Application Architecture Guide》(Microsoft Press, 2009) 包含了关于移动 App 设计和架构的有用信息。Lee 的书 (《Mobile Applications: Architecture, Design, and Development》, Prentice Hall, 2004) 讨论了平板设备的体系结构设计。Esposito (《Architecting Mobile Solutions for the Enterprise》, Microsoft Press, 2012) 描述了通过建立一个有效的移动网站开发跨平台应用的过程。Garbinato (《Middleware for Network Eccentric and Mobile Applications》, Springer, 2009) 编写的书讨论了移动中间件。

有许多着眼于特定平台的关于移动 App 编程的书。Firtman (《Programming the Mobile Web》, O'Reilly, 2010), Mednieke (《Programming Android》, O'Reilly, 2011) 或 Lee (《Test-Driven iOS Development》, Addison-Wesley, 2012) 的书都具有代表性。

可以在网上获得移动 App 设计方面的大量信息, 最新的参考文献可以在 SEPA 网站 www.mhhe.com/pressman 的“software engineering resources”下找到。

质量管理

本书的这一部分将学习用来管理和控制软件质量的原理、概念和技术。在后面几章中会涉及下列问题：

- 高质量软件的一般特征是什么？
- 如何评审质量？如何有效地进行质量评审？
- 什么是软件质量保证？
- 软件测试需要应用什么策略？
- 使用什么方法才能设计出有效的测试用例？
- 有没有确保软件正确性的可行方法？
- 如何管理和控制软件开发过程中经常发生的变更？
- 使用什么标准和尺度评估需求模型、设计模型、源代码以及测试用例的质量？

回答了这些问题，就为保证生产出高质量软件做好了准备。

质量概念

要点浏览

概念：究竟什么是软件质量？答案不是想象中的那样容易，当你看到它时你知道质量是什么，可是，它却不可捉摸，难以定义。但是对于计算机软件，质量是必须定义的，这正是本章要讲的。

人员：软件过程所涉及的每个人（软件工程师、经理和所有利益相关者）都对质量负有责任。

重要性：你可以把事情做好，或者再做一遍。如果软件团队在所有软件工程活动中强调质量，就可以减少很多必需的返工，结果是降低了成本，更为重要的是

缩短了上市时间。

步骤：为实现高质量软件，必须具备四项活动：已验证的软件工程过程和实践，扎实的项目管理，全面的质量控制，具有质量保证基础设施。

工作产品：满足客户需要的、准确而可靠地运行的、为所有使用者提供价值的软件。

质量保证措施：通过检查所有质量控制活动的结果来跟踪质量，通过在交付前检查错误，在发布到现场后检查缺陷来衡量质量。

随着软件日益融入人们生活的方方面面，提高软件质量的鼓声真的要播响了。截至 20 世纪 90 年代，大公司认识到由于软件达不到承诺的特性和功能，每年浪费的钱财多达数十亿美元。更严重的是，政府和产业界开始日益担心严重的软件缺陷有可能使重要的基础设施陷入瘫痪，从而使花费超过数百亿美元。世纪之交，CIO 杂志一篇标题为“停止每年浪费 780 亿美元”的文章，对“美国企业在不能如预期那样工作的软件上花费数十亿美元”[Lev01]这一事实表示遗憾。InformationWeek[Ric01]也表达了同样的担忧：

市场研究公司的 Standish Group 谈到，尽管意愿良好，但有缺陷的代码仍然是软件工业的幽灵，计算机系统的故障时间高达 45%，美国公司去年在丧失的生产率和修补上大约花费了一千亿美元，这还不包括失去那些怒气冲冲的客户的代价。因为 IT 企业依赖基础软件包来开发应用，所以糟糕代码也能损毁定制的应用。

多差的软件才是劣质软件呢？人们对此的定义是不同的，但是专家认为，只要每 1000 行代码有 3 或 4 处缺陷就能使程序执行得很差，大多数程序员每写 10 行代码大约注入一个错误，许多商业产品有数百万行代码，软件经销商至少将开发预算的一半花费在了测试时修

关键概念

质量的成本
足够好
责任
管理活动
质量
质量困境
质量维度
质量因素
量化观点
风险
安全

改错误上。

2005年, ComputerWorld[Hil05]遗憾地表示, 劣质软件惹恼了几乎所有使用计算机的组织机构, 在计算机发生故障期间造成了工作时间损失、数据丢失或毁坏、销售时机丧失、IT支持与维护费用高昂, 以及客户满意度低等后果。一年后, InfoWorld[Fos06]以“软件质量的可悲状况”作为主题书写了报告, 报告称质量问题依然没有任何改观。随着人们对软件质量的日益重视, 一项就 100000 位白领专业人士的调查 [Rog12] 显示软件质量工程师是“全美最幸福的工作者”!

现如今, 软件质量仍然是个问题, 但是应该责备谁? 客户责备开发人员, 认为草率的做法导致低质量的软件。开发人员责备客户(和其他项目利益相关者), 认为不合理的交工日期以及连续不断的变更使开发人员在还没有完全验证时就交付了软件。谁说的对? 都对, 这正是问题所在。本章把软件质量作为一个概念, 考查在软件工程实践中为什么软件质量值得认真考虑。

19.1 什么是质量

Robert Persig[Per74]在他的神秘之书《Zen and the Art of MotorCycle Maintenance》中我们就称为“质量”的东西发表了看法:

质量……你知道它是什么, 也不知道它是什么。这样说是自相矛盾的, 但没有比这更好的说法了——这样说更有质量。但是当要试图说明质量是什么时, 除了我们知道的这些之外, 总是所知了了! 没有什么好谈论的, 但是如果不能说明质量是什么, 又怎能知道质量是什么, 或者又怎能知道质量甚至是否存在呢? 如果没有人知道质量是什么, 那么所有实际用途就根本不存在, 但是所有实际用途确实是存在的。质量等级依据其他哪些东西来划分? 为什么人们将机会给某些东西而把其他东西扔到垃圾堆? 显而易见某些东西好于其他东西……但是, 好在什么地方呢? ……所以就任凭金属的轮子一圈又一圈地转动, 而找不到摩擦力在哪里。质量到底是什么? 什么是质量?

的确, 什么是质量?

在更为实用的层面上, 哈佛商学院的 David Garvin[Gar84]给出了建议: “质量是一个复杂多面的概念”, 可以从 5 个不同的观点来描述。先验论观点(如 Persig)认为质量是马上就能识别的东西, 却不能清楚地定义。用户观点是从最终用户的具体目标来考虑的。如果产品达到这些目标, 就是有质量的。制造商观点是从产品原始规格说明的角度来定义质量, 如果产品符合规格说明, 就是有质量的。产品观点认为质量是产品的固有属性(比如功能和特性)。最后, 基于价值的观点根据客户愿意为产品支付多少钱来评测质量。实际上, 质量涵盖所有这些观点, 或者更多。

设计质量是指设计师赋予产品的特性。材料等级、公差和性能等规格说明决定了设计质量。如果产品是按照规格说明书制造的, 那么使用较高等级的材料, 规定更严格的公差和更高级别的性能, 产品的设计质量就能提高。

在软件开发中, 设计质量包括设计满足需求模型规定的功能和特性的程度。符合质量关注的是实现遵从设计的程度以及所得到的系统满足需求和性能目标的程度。

但是, 设计质量和符合质量是软件工程师必须考虑的唯一问题吗?

提问 在看待质量方面有哪些不同的方法?

引述 人们记不住你做一项工作有多快——但总能记住你做得有多好。

Howard Newton

Robert Glass [Gla98] 认为它们之间比较“直观的”关系符合下面的公式：

用户满意度 = 合格的产品 + 好的质量 + 按预算和进度安排交付

总之，Glass 认为质量是重要的。但是，如果用户不满意，其他任何事情也就都不重要了。DeMarco [DeM98] 同意这个观点，他认为：“产品的质量是一个函数，该函数确定了它在多大程度上使这个世界变得更好。”这个质量观点的意思就是：如果一个软件产品能给最终用户带来实质性的益处，那么他们可能会心甘情愿地忍受偶尔的可靠性或性能问题。

19.2 软件质量

高质量的软件是一个重要目标，即使最疲倦的软件开发人员也会同意这一点。但是，如何定义软件质量呢？在最一般的意义上，软件质量可以这样定义：在一定程度上应用有效的软件过程，创造有用的产品，为生产者和使用者提供明显的价值。^①

提问 如何为软件质量下一个最好的定义？

毫无疑问，对上述定义可以进行修改、扩展以及无休止的讨论。针对本书的论题来说，该定义强调了以下三个重要的方面：

引述 作为卓越的代名词，一些人并不能适应需要杰出素质的环境。

Steve Jobs

1. 有效的软件过程为生产高质量的软件产品奠定了基础。过程的管理方面所做的工作是检验和平衡，以避免项目混乱（低质量的关键因素）。软件工程实践允许开发人员分析问题、设计可靠的解决方案，这些都是生产高质量软件的关键所在。最后，诸如变更管理和技术评审等普适性活动与其他部分的软件工程活动密切相关。
2. 有用的产品是指交付最终用户要求的内容、功能和特征，但最重要的是，以可靠、无误的方式交付这些东西。有用的产品总是满足利益相关者明确提出的那些需求，另外，也要满足一些高质量软件应有的隐性需求（例如易用性）。
3. 通过为软件产品的生产者和使用者增值，高质量软件为软件组织和最终用户群体带来了收益。软件组织获益是因为高质量的软件在维护、改错及客户支持方面的工作量都降低了，从而使软件工程师减少了返工，将更多的时间花费在开发新的应用上，软件组织因此而获得增值。用户群体也得到增值，因为应用所提供的有用的能力在某种程度上加快了一些业务流程。最后的结果是：（1）软件产品的收入增加；（2）当应用可支持业务流程时，收益更好；（3）提高了信息可获得性，这对商业来讲是至关重要的。

19.2.1 Garvin 的质量维度

David Garvin [Gar87] 建议采取多维的观点考虑质量，包括从符合性评估到抽象的（美学）观点。尽管 Garvin 的 8 个质量维度没有专门为软件制定，但考虑软件质量时依然可以使用。

建议 当应用采用了 Garvin 的质量维度时，可以使用雷达图提供每个 Garvin 质量维度的可视化表现形式。

性能质量。 软件是否交付了所有的内容、功能和特性？这些内容、功能和特性在某种程度上是需求模型所规定的一部分，可以为最终用户提供价值。

特性质量。 软件是否提供了使第一次使用的最终用户感到惊喜的特性？

可靠性。 软件是否无误地提供了所有的特性和能力，当需要使用该软件时，它是否是可

^① 该定义节选自 [Bes04]，取代该书前几版中出现的更面向生产的观点。

用的,是否无错地提供了功能?

符合性。软件是否遵从本地的和外部的与应用领域相关的软件标准,是否遵循了事实存在的设计惯例和编码惯例?例如,对于菜单选择和数据输入等用户界面的设计是否符合人们已接受的设计规则?

耐久性。是否能够对软件进行维护(变更)或改正(改错),而不会粗心大意地产生意想不到的副作用?随着时间的推移,变更会使错误率或可靠性变得更糟吗?

适用性。软件能在可接受的短时期内完成维护(变更)和改正(改错)吗?技术支持人员能得到所需的所有信息以进行变更和修正缺陷吗? Douglas Adams[Ada93] 挖苦地评论道:“可能发生故障的东西与不可能发生故障的东西之间的差别是,当前者发生了故障时,通常是不可能发现和补救的”。

审美。毫无疑问,关于什么是美的,我们每个人有着不同的、非常主观的看法。可是,我们中的大多数都同意美的东西具有某种优雅、特有的流畅和醒目的外在,这些都是很难量化的,但显然是不可缺少的,美的软件具有这些特征。

感知。在某些情况下,一些偏见将影响人们对质量的感知。例如,有人给你介绍了一款软件产品,该软件产品是由过去曾经生产过低质产品的厂家生产的,你的自我保护意识将会增加,你对于当前软件产品质量的感知力可能受到负面影响。类似地,如果厂家有极好的声誉,你将能感觉到好的质量,甚至在实际质量并不真的如此时,你也会这样想。

Garvin 的质量维度提供了对软件质量的“软”评判。这些维度中的多数(不是所有)只能主观地考虑。正因如此,也需要一套“硬”的质量因素,这些因素可以宽泛地分成两组:(1)可以直接测量的因素(例如,测试时发现的缺陷数);(2)只能间接测量的因素(例如,可用性或可维护性)。在任何情况下,必须进行测量,应把软件和一些基准数据进行比较来确定质量。

19.2.2 McCall 的质量因素

McCall、Richards 和 Walters[McC77] 提出了影响软件质量因素的一种有用的分类。这些软件质量因素侧重于软件产品的三个重要方面:操作特性、承受变更的能力以及对新环境的适应能力,如图 19-1 所示。

针对图 19-1 中所提到的因素,McCall 及他的同事提供了如下描述:

正确性。程序满足其需求规格说明和完成用户任务目标的程度。

可靠性。期望程序以所要求的精度完成其预期功能的程度。需要提醒大家注意的是,还有更完整的可靠性定义(第 21 章)。

效率。程序完成其功能所需的计算资源和代码的数量。

完整性。对未授权的人员访问软件或数据的可控程度。

易用性。对程序进行学习、操作、准备输入和解释输出所需要的工作量。

可维护性。查出和修复程序中的一个错误所需要的工作量。(这是一个

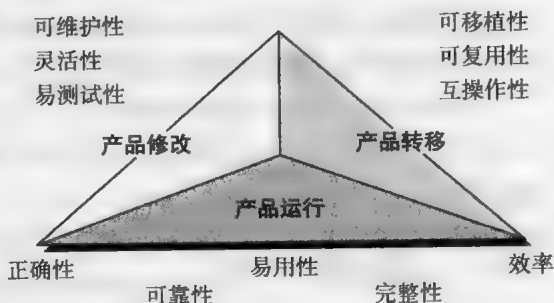


图 19-1 McCall 的软件质量因素

描述 满足进度计划的喜悦已经被遗忘很久之后,低质量所带来的痛苦依然挥之不去。

Karl Weigers

非常受限的定义。)

灵活性。修改一个运行的程序所需的工作量。

易测试性。测试程序以确保它能完成预期功能所需要的工作量。

可移植性。将程序从一个硬件和软件系统环境移植到另一个环境所需要的工作量。

可复用性。程序(或程序的一部分)可以在另一个应用中使用的程度。这与程序所执行功能的封装和范围有关。

互操作性。将一个系统连接到另一系统所需要的工作量。

要想求得这些质量因素的直接测度[⊖]是困难的,且在有些情况下是不可能的。事实上,由 McCall 等人定义的度量仅能间接地测量。不过,使用这些因素评估应用的质量可以真实地反映软件的质量。

417

19.2.3 ISO 9126 质量因素

ISO 9126 国际标准的制定是试图标识计算机软件的质量属性。这个标准标识了 6 个关键的质量属性。

功能性。软件满足已确定要求的程度,由以下子属性表征:适合性、准确性、互操作性、依从性和安全性。

可靠性。软件可用的时间长度,由以下子属性表征:成熟性、容错性和易恢复性。

易用性。软件容易使用的程度,由以下子属性表征:易理解性、易学习性和易操作性。

效率。软件优化使用系统资源的程度,由以下子属性表征:时间特性和资源利用特性。

可维护性。软件易于修复的程度,由以下子属性表征:易分析性、易改变性、稳定性和易测试性。

可移植性。软件可以从一个环境移植到另一个环境的容易程度,由以下子属性表征:适应性、易安装性、符合性和易替换性。

与前面几小节讨论的软件质量因素一样,ISO 9126 中的质量因素不一定有助于直接测量。然而,它们确实为间接测量提供了有价值的基础,并为评估系统质量提供了一个优秀的检查单。

建议 尽管为这里提到的质量因素制定量化测量是很诱人的,但你也可以创建一个简单的属性清单来提供显示质量因素的可靠指标。

19.2.4 定向质量因素

19.2.1 节和 19.2.2 节提出的质量维度和因素关注软件的整体,可以用其作为表征应用质量的一般性指标。软件团队可以提出一套质量特征和相关的问题以调查软件满足每个质量因素的程度[⊖]。例如:McCall 把易用性看作重要的质量因素。当要求评审用户界面和评估易用性时,该如何进行?可能要从 McCall 提出的子属性——易理解性、易学习性和易操作性开始,但是在实用的意义上,这些子属性表示什么意思呢?

引述 当从业人员想将事情做正确或做得更好时,任何活动都将变得富有创造性。

John Updike

⊖ 直接测度意味着存在一个简单可计算的值。该值为被考察的属性提供直接的指标。例如,程序的“规模”可以通过计算代码的行数直接测量。

⊖ 这些特征和问题将作为软件评审的一部分来对待(第 20 章)。

418

为了进行评价,需要说清楚界面具体的、可测量的(或至少是可识别的)属性。例如下面这些属性 [Bro03]。

直觉。界面遵照预期使用模式的程度,使得即使是新手也能不经过专门培训而开始使用。

- 界面布局易于理解吗?
- 界面操作容易找到和上手吗?
- 界面使用了可识别的隐喻吗?
- 输入的安排可尽量少敲击键盘和点击鼠标吗?
- 界面符合三个重要原则吗(第15章)?
- 美学的运用有助于理解和使用吗?

效率。定位或初步了解操作和信息的程度。

- 界面的布局 and 风格可以使用户有效地找到操作和信息吗?
- 一连串的操作(或数据输入)可以用简单动作实现吗?
- 输出的数据和显示的内容是否能立即被理解?
- 分层操作是否组织得能使用户完成某项工作所需导航的深度最小?

健壮性。软件处理有错的输入数据或不恰当的用户交互的程度。

- 如果输入了规定边界上的数据或恰好在规定边界外的数据,软件能识别出错误吗?更为重要的是,软件还能继续运行而不出错或性能不下降吗?
- 界面能识别出常见的可识别错误或操作错误,并能清晰地指导用户回到正确的轨道上来吗?
- 若发现了错误的情况(与软件功能有关),界面是否提供了有用的诊断和指导?

丰富性。界面提供丰富特征集的程度。

- 界面是否能按照用户的特定要求进行定制?
- 界面是否提供宏操作以使用户将单个的行为或命令当作一连串的常用操作?

当界面设计展开后,软件团队将评审设计原型,询问他们所关注的问题。如果对这些问题的大多数回答是“肯定的”,用户界面就具备了高质量。应该为每个待评估的质量因素开

419

19.2.5 过渡到量化观点

前面几节讨论了一组测量软件质量的定性因素。软件界也力图开发软件质量的精确的测量,但有时又会为活动的主观性而受挫。Cavano 和 McCall[Cav78] 讨论了这种情形:

质量评定是日常事件(葡萄酒品尝比赛、运动赛事(如体操)、智力竞赛等)中的一个关键因素。在这些情况下,质量是以最基本、最直接的方式来判断的:在相同的条件和预先决定的概念下将对象进行并列对比。葡萄酒的质量可以根据清澈度、颜色、酒花和味道等来判断。然而,这种类型的判断是很主观的,最终的结果必须由专家给出。

主观性和特殊性也适用于软件质量的评定。为了帮助解决这个问题,需要对软件质量有一个更精确的定义,同样,为了客观分析,需要产生软件质量的定量测量方法……既然没有这种事物的绝对知识,就不要期望精确测量软件质量,因为每一种测量都是部分地不完美的。Jacob

建议 当面临质量困境时(每个人都会在某个时期面对它),尽力取得平衡——用足够的工作量去开发质量可接受的产品,而不是将项目葬送掉。

Bronkowski 这样描述知识的自相矛盾现象：“年复一年，我们设计更精准的仪器用以更精细地观察自然界，可是当我们留心这些观察数据，却很不愉快地发现这些数据依然模糊时，我们感到它们还和过去一样不确定。”

第 30 章将提出一组可应用于软件质量定量评估的软件度量。在所有的情况下，这些度量都表示间接的测度。也就是说，我们从不真正测量质量，而是测量质量的一些表现。复杂因素在于所测量的变量和软件质量间的精确关系。

19.3 软件质量困境

在网上发布的一篇访谈 [Ven03] 中，Bertrand Meyer 这样论述我所谓的质量困境：

如果生产了一个存在严重质量问题的软件系统，你将受到损失，因为没有人想去购买。另一方面，如果你花费无限的时间、极大的工作量和高额的资金来开发一个绝对完美的软件，那么完成该软件将花费很长的时间，生产成本是极其高昂的，甚至会破产。要么错过了市场机会，要么几乎耗尽所有的资源。所以企业界的人努力达到奇妙的中间状态：一方面，产品要足够好，不会立即被抛弃（比如在评估期）；另一方面，又不是那么完美，不需花费太长时间和太多成本。

软件工程师应该努力生产高质量的系统，在这一过程中如果能采用有效的方法就更好了。但是，Meyer 所讨论的情况是现实的，甚至对于最好的软件工程组织，这种情况也表明了一种两难的困境。

建议 虽然开发这里提到的质量因素的量化测量是临时的，但还是可以创建一个简单的属性检查表，这些属性可以提供表征质量因素的可信指示。

420

19.3.1 “足够好”的软件

坦率地说，如果我们准备接受 Meyer 的观点，那么生产“足够好”软件是可接受的吗？对于这个问题的答案只能是“肯定的”，因为大型软件公司每天都在这么做。这些大公司生产带有已知缺陷的软件，并发布给大量的最终用户。他们认识到，1.0 版提供的一些功能和特性达不到最高质量，并计划在 2.0 版改进。他们这样做时，知道有些客户会抱怨，但他们认识到上市时间胜过更好的质量，只要交付的产品“足够好”。

到底什么是“足够好”？足够好的软件提供用户期望的高质量功能和特性，但同时也提供了其他更多的包含已知错误的难解的或特殊的功能和特性。软件供应商希望广大的最终用户忽视错误，因为他们对其他的应用功能是如此满意。

这种想法可能引起许多读者的共鸣。如果你是其中之一，我只能请你考虑一些论据来反对“足够好”。

诚然，“足够好”可能在某些应用领域和几个主要的软件公司起作用。毕竟，如果一家公司有庞大的营销预算，并能够说服足够多的人购买 1.0 版本，那么该公司已经成功地锁定了这些用户。正如前面所指出的，可以认为，公司将在以后的版本提高产品质量。通过提供足够好的 1.0 版，公司垄断了市场。

如果你所在的是一个公司，就要警惕这一观念，当你交付一个足够好的（有缺陷的）产品时，你是冒着永久损害公司声誉的风险。你可能再也没有机会提供 2.0 版本了，因为异口同声的不良评论可能会导致销售暴跌乃至公司关门。

如果你工作在某个应用领域（如实时嵌入式软件），或者你构建的是与硬件集成的应用软件（如汽车软件、电信软件），那么一旦交付了带有已知错误的软件（也许是一时疏忽），

就可能使公司处于代价昂贵的诉讼之中。在某些情况下，甚至可能是刑事犯罪，没有人想要足够好的飞机航空电子系统软件！

因此，如果你认为“足够好”是一个可以解决软件质量问题的捷径，那么要谨慎行事。“足够好”可以起作用，但只是对于少数几个公司，而且只是在有限的几个应用领域。^①

[421]

19.3.2 质量的成本

关于软件成本有这样的争论：我们知道，质量是重要的，但是花费时间和金钱——花费太多的时间和金钱才能达到我们实际需要的软件质量水平。表面上看，这种说法似乎是合理的（见本节前面 Meyer 的评论）。毫无疑问，质量好是有成本的，但质量差也有成本——不仅是对必须忍受缺陷软件的最终用户，而且是对已经开发且必须维护该软件的软件组织。真正的问题是：我们应该担心哪些成本？要回答这个问题，你必须既要了解实现质量的成本，又要了解低质量软件的成本。

质量成本包括追求质量过程中或在履行质量有关的活动中引起的费用以及质量不佳引起的下游费用等所有费用。为了解这些费用，一个组织必须收集度量数据，为目前的质量成本提供一个基准，找到降低这些成本的机会，并提供一个规范化的对比依据。质量成本可分为预防成本、评估成本和失效成本。

预防成本包括：（1）计划和协调所有质量控制和质量保证所需管理活动的成本；（2）为开发完整的需求模型和设计模型所增加的技术活动的成本；（3）测试计划的成本；（4）与这些活动有关的所有培训成本。

评估成本包括为深入了解产品“第一次通过”每个过程的条件而进行的活动。评估成本的例子包括：（1）对软件工作产品进行技术评审（第 20 章）的成本；（2）数据收集和度量估算（第 30 章）的成本；（3）测试和调试（第 22～26 章）的成本。

失效成本是那些在将产品交付客户之前若没有出现错误就不会发生的费用。失效成本可分为内部失效成本和外部失效成本。内部失效成本发生在软件发布之前发现错误时，内部失效成本包括：（1）为纠正错误进行返工（修复）所需的成本；（2）返工时无意中产生副作用，必须对副作用加以缓解而发生的成本；（3）组织为评估失效的模型而收集质量数据，由此发生的相关成本。外部失效成本是在产品已经发布给客户之后发现了缺陷时的相关成本。外部成本的例子包括：解决投诉，产品退货和更换，帮助作业支持，以及与保修工作相关的人力成本。不良的声誉和由此产生的业务损失是另一个外部失效成本，这是很难量化但非常现实的。生产了低质量的软件产品时，不好的事情就要发生。

建议 不要担心预防成本显著增加。放心，你的投资将会获得良好的回报。

引述 把事情做对比解释为什么做错花费的时间更少。

H. W. Longfellow

[422]

在对拒绝考虑外部失效成本的软件开发者的控告书中，Cem Kaner [Kan95] 是这样说的：

许多外部失效成本（如声誉的损失）都难以量化，因此许多企业在计算其成本效益的权衡时忽视了这些成本。还有一些外部失效成本可以降低（例如，通过提供更便宜、质量更低的产品，售后支持，或向客户收取支持费用），这些都不会增加用户的满意度。质量工程师靠忽视使用不良产品的客户成本，鼓励做出相关的质量决策，这种决策只会欺骗客户，而不会使客户高兴。

① 关于“足够好”软件的优缺点的有价值的探讨请参阅 [Bre02]。

正如预料的那样,当我们从预防到检查内部失效成本和外部失效成本时,找到并修复错误或缺陷的相关成本会急剧增加。根据 Boehm 和 Basili 收集的数据 [Boe01b] 以及 Cigital Inc[Cig07] 的阐述,图 19-2 说明了这一现象。

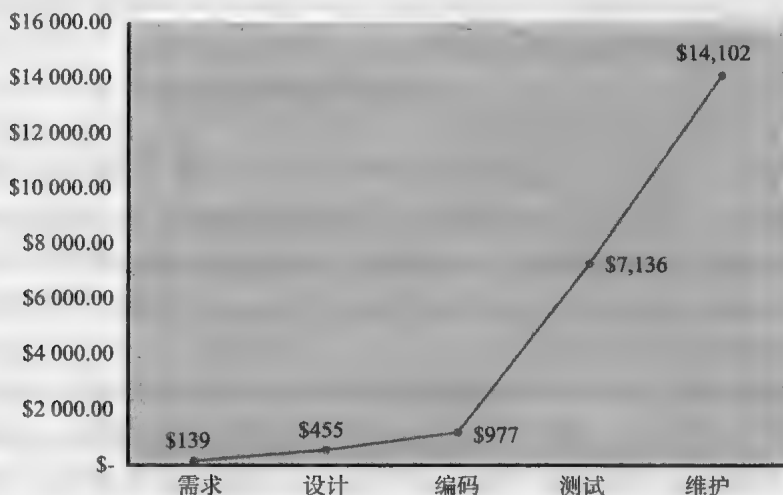


图 19-2 改正错误和缺陷的相对成本 [Boe01b]

在代码生成期纠正缺陷的行业平均成本是每个错误大约 977 美元,而在系统测试期,纠正同样错误的行业平均成本是每个错误 7136 美元。[Cig07] 认为,一个大型应用程序在编码期会引入 200 个错误。

根据行业平均水平的数据,在编码阶段发现和纠正缺陷的成本是每个缺陷 977 美元,因此,在本阶段纠正这 200 个“关键”缺陷的总费用大约是 195400 美元 (200×977 美元)。

行业平均水平数据显示,在系统测试阶段发现和纠正缺陷的代价是每个缺陷 7136 美元。在这种情况下,假定该系统测试阶段发现大约 50 个关键缺陷(或者 Cigital 在编码阶段只发现了这些缺陷的 25%),发现和解决这些缺陷的代价将是大约 356800 美元 (50×7136 美元)。这将导致 150 处关键错误未被发现和矫正。在维护阶段发现和解决这些遗留的 150 个缺陷的代价将是 2115300 美元 (150×14102 美元)。因此,在编码阶段之后发现和解决这 200 个缺陷的代价将是 2472100 美元 (2115300 美元 + 356 800 美元)。

即使软件组织所花费的是行业平均水平的一半(大多数企业尚不知道他们的成本!),与早期的质量控制和保证活动(进行需求分析和设计)有关的成本节约也是不得不做的。

SafeHome 质量问题

[场景] Doug Miller 的办公室, SafeHome 软件项目开始。

[人物] Doug Miller (SafeHome 软件工程项目团队经理) 和产品软件工程团队的其他成员。

[对话]

Doug: 我正在看一份关于软件缺陷修改成

本的行业报告,缺陷修改成本令人警醒。

Jamie: 我们已经准备好为每一个功能需求开发测试用例。

Doug: 好的,我注意到修复在测试期间发现的一处缺陷要比在编码期间发现并修复一处缺陷花费八倍的工作量。

Vinod: 我们正使用结对编程法,所以我们

应能捕获编码期间的大多数缺陷。

Doug: 我认为你没有抓住重点, 软件质量不只是简单地去除编码错误。我们需要关注项目质量目标, 保证不断变更的软件产品满足这些目标。

Jamie: 您的意思是可用性、安全性和可靠性那些问题吗?

Doug: 是的, 我们需要在软件过程中加入检查机制, 以监视过程是朝着质量目标方

向的。

Vinod: 难道我们不能在完成第一个原型之后进行质量检查吗?

Doug: 恐怕不能, 我们必须在项目早期建立质量文化。

Vinod: 您要我们做什么, Doug?

Doug: 我认为需要寻找一种使我们能监视 SafeHome 产品质量的技术, 让我们考虑一下, 明天再讨论这个话题。

19.3.3 风险

本书的第1章写道: “人们拿自己的工作、自己的舒适、自己的安全、自己的娱乐、自己的决定以及自己的生命在计算机软件上下赌注。最好这是正确的。”这意味着, 低质量的软件为开发商和最终用户都增加了风险。前面讨论了这些风险(成本)中的一个, 但设计和实现低劣的应用所带来的损失并不总是限于美元和时间, 一个极端的例子[Gag04]可能有助于说明这一点。

整个2000年11月份, 在巴拿马的一家医院, 28位病人在治疗多种癌症的过程中受到了过量的伽马射线照射。此后数月内, 其中5例死于辐射病, 15人发展成严重的并发症。是什么造成了这一悲剧? 这是因为医院的技术人员对一家美国公司开发的软件包进行了修改, 以计算每位病人的辐射剂量的变更值。

为了获取额外的软件功能, 三位巴拿马医疗物理学家“调整”了软件, 他们被指控犯有二级谋杀罪。同时, 这家美国软件公司正在两个国家面临着严重的诉讼。Gage 和 McCormick 评论道:

这不是一个讲给医疗技术人员的警示故事, 尽管由于误解或误用了技术, 他们要为了免于牢狱之灾而据理力争。这也不是一个关于人类如何受到伤害的故事, 或者更糟的是被设计不良或说明不清的软件伤害, 尽管这样的例子比比皆是。这是任何一个计算机程序设计者都应当铭记的教训: 软件质量问题很重要, 不管软件是嵌入汽车引擎中、工厂里的机械手臂中, 还是嵌入医院的治疗设备中, 这些应用必须做到万无一失, 低劣部署的代码可以杀人。

质量低劣导致风险, 其中一些风险会非常严重。

19.3.4 疏忽和责任

这种情况太常见了。政府或企业雇佣一个较大的软件开发商或咨询公司来分析需求, 然后设计和创建一个基于软件的“系统”, 用以支撑某个重大的活动。系统可能支持主要的企业功能(例如, 养老金管理), 或某项政府职能(例如, 卫生保健管理或国土安全)。

工作始于双方良好的意愿, 但是到了系统交付时, 情况已变得糟糕, 系统延期, 未能提供预期的特性和功能, 而且易出错, 不能得到客户的认可, 接下来就要打官司。

在大多数情况下, 顾客称开发商马虎大意(已带到了软件实践中), 因此拒绝付款。而开发商则常常声称, 顾客一再改变其要求, 并在其他方面破坏了开发伙伴关系。无论是哪一种情况, 交付系统的质量都会有问题。

19.3.5 质量和安全

随着基于 Web 的系统和移动系统重要性的增加，应用的安全性已变得日益重要。简而言之，没有表现出高质量的软件比较容易被攻击。因此，低质量的软件会间接地增加安全风险，随之而来的是费用和问题。

在 ComputerWorld 的一篇访谈中，作者和安全专家 Gary McGraw 这样评论 [Wil05]：

425

软件安全与质量息息相关。必须一开始就在设计、构建、测试、编码阶段以及在整个软件生命周期（过程）中考虑安全性、可靠性、可得性、可信性。即使是已认识到软件安全问题的人也会主要关注生命周期的晚些阶段。越早发现软件问题越好。有两种类型的软件问题，一种是隐藏的错误的实现的问题。另一种是软件缺陷，这是设计中的构建问题。人们对错误关注太多，却对缺陷关注不够。

要构造安全的系统，就必须注重质量，并必须在设计时开始关注。本书第二部分讨论的概念和方法可以导出减少“缺陷”的软件架构。我们将在第 27 章更详细地讨论软件安全工程。

19.3.6 管理活动的影响

软件质量受管理决策的影响往往和受技术决策的影响是一样的。即使最好的软件工程实践也能被糟糕的商业决策和有问题的项目管理活动破坏。

本书第四部分讨论软件过程环境下的项目管理。每个项目任务开始时，项目领导人都要作决策，这些决策可能对产品质量有重大影响。

估算决策。在确定交付日期和制定总预算之前，给软件团队提供项目估算数据是很少见的。反而是团队进行了“健康检查”，以确保交付日期和里程碑是合理的。在许多情况下，存在着巨大的上市时间压力，迫使软件团队接受不现实的交付日期。结果，由于抄了近路，可以获得更高质量软件的活动被忽略掉了，产品质量受到损害。如果交付日期是不合理的，那么坚持立场就是重要的。这就解释了为什么你需要更多的时间，或者也可以建议在指定的时间交付一个（高质量的）功能子集。

426

进度安排决策。一旦建立了软件项目时间表（第 34 章），就会按照依赖性安排任务的先后顺序。例如，由于 A 构件依赖于 B、C 和 D 构件中的处理，因此直到 B、C 和 D 构件完全测试后，才能安排 A 构件进行测试。项目计划将反映这一点。但是，如果时间很紧，为了做进一步的关键测试，A 必须是可用的。在这种情况下，可能会决定在没有其附属构件（这些附属构件的运行要稍落后于时间表）的情况下测试 A，这样对于交付前必须完成的其他测试，就可以使用 A 了，毕竟，期限正在逼近。因此，A 可能有隐藏的缺陷，只有晚些时候才能发现，质量会受到影响。

面向风险的决策。风险管理（第 35 章）是成功软件项目的关键特性之一。必须知道哪里可能会出问题，并建立一项如果确实出问题时的应急计划。太多的软件团队喜欢盲目乐观，在什么都不会出问题的假设下建立开发计划。更糟的是，他们没有办法处理真的出了差错的事情。结果，当风险变成现实后，便会一片混乱，并且随着疯狂程度的上升，质量水平必然下降。

用 Meskimen 定律能最好地概括软件质量面临的困境——从来没有时间做好，但总是有时间再做一遍。我的建议是，花点时间把事情做好，这几乎从来都不是错误的决定。

19.4 实现软件质量

良好的软件质量不会自己出现，它是良好的项目管理和扎实的软件工程实践的结果。帮助软件团队实现高质量软件的四大管理和实践活动是：软件工程方法、项目管理技术、质量控制活动以及软件质量保证。

19.4.1 软件工程方法

如果希望建立高质量的软件，就必须理解要解决的问题。还须能够创建一个符合问题的设计，该设计同时还要具备一些性质，这些性质可以使我们得到具有 19.2 节讨论过的质量维度和因素的软件。

本书第二部分提出了一系列概念和方法，可帮助我们获得对问题合理完整的理解和综合性设计，从而为构建活动建立了坚实的基础。如果应用这些概念，并采取适当的分析和设计方法，那么创建高质量软件的可能性将大大提高。

19.4.2 项目管理技术

在 19.3.6 节已经讨论了不良管理决策对软件质量的影响。其中的含义是明确的：如果（1）项目经理使用估算以确认交付日期是可以达到的；（2）进度依赖关系是清楚的，团队能够抵抗走捷径的诱惑；（3）进行了风险规划，这样出了问题就不会引起混乱，软件质量将受到积极的影响。

提问 需要做些什么以对质量产生积极的影响？

此外，项目计划应该包括明确的质量管理和变更管理技术。导致良好项目管理实践的技术将在本书第四部分讨论。

19.4.3 质量控制

质量控制包括一套软件工程活动，以帮助确保每个工作产品符合其质量目标。评审模型以确保它们是完整的和一致的。检查代码，以便在测试开始前发现和纠正错误。应用一系列的测试步骤以发现逻辑处理、数据处理以及接口通信中的错误。当这些工作成果中的任何一个不符合质量目标时，测量和反馈的结合使用使软件团队可以调整软件过程。本书第三部分的其余章节对质量控制活动进行了详细的讨论。

提问 软件质量控制是什么？

427

19.4.4 质量保证

质量保证建立基础设施，以支持坚实的软件工程方法，合理的项目管理和质量控制活动——如果你打算建立高品质软件，那么所有这些都是关键活动。此外，质量保证还包含一组审核和报告功能，用以评估质量控制活动的有效性和完整性。质量保证的目标是为管理人员和技术人员提供所需的数据，以了解产品的质量状况，从而理解和确信实现产品质量的活动在起作用。当然，如果质量保证中提供的数据出现了问题，那么处理问题和使用必要的资源来解决质量问题是管理人员的职责。软件质量保证将在第 21 章详细论述。

网络资源 有关软件质量保证的有用资料可以在下列站点找到：
[www.niwotridge.com/Resources/PM-SWEResources/Software Quality Assurance.htm](http://www.niwotridge.com/Resources/PM-SWEResources/Software%20Quality%20Assurance.htm)

19.5 小结

软件正在融入我们日常生活的各个方面，人们对于软件系统质量的关注也逐渐多起来。但是很难给出软件质量的一个全面描述。在这一章，质量被定义为一个有效的软件过程，用来在一定程度上创造有用的产品，为那些生产者和使用者提供适度的价值。

这些年来，提出了各种各样的软件质量度量和因素，都试图定义一组属性，如果可以实现，那么我们将实现较高的软件质量。McCall 的质量因素和 ISO 9126 的质量因素建立了很多特性（如可靠性、易用性、维护性、功能性和可移植性）作为质量存在的指标。

每个软件组织都面临软件质量困境。从本质上说，每个人都希望建立高质量的系统，但生产“完美”软件所需的时间和工作量在市场主导的世界里根本无法达到。这样问题就转化为，我们是否应该生产“足够好”的软件？虽然许多公司是这样做的，但是这样做有很大的负面影响，必须加以考虑。

不管选择什么方法，质量都是有成本的，质量成本可以从预防、评估和失效方面来说。预防成本包括所有将预防缺陷放在首位的软件工程活动。评估成本是与评估软件工作产品以确定其质量的活动有关的成本。失效成本包括失效的内部代价和低劣质量造成的外部影响。

软件质量是通过软件工程方法、扎实的管理措施和全面质量控制的应用而实现的——所有这些是靠软件质量保证基础设施支持的。后续章节将详细讨论质量控制和质量保证。

习题与思考题

- 19.1 描述在申请学校之前你将怎样评估一所大学的质量。哪些因素重要？哪些因素关键？
- 19.2 Garvin[Gar84] 描述了质量的 5 种不同的观点，用一个或多个您熟悉的知名电子产品为每个观点举一个例子。
- 19.3 使用 19.2 节提出的软件质量定义，不使用有效的过程来创建能提供明显价值的有用产品，你认为可能吗？解释你的答案。
- 19.4 为 19.2.1 节介绍的 Garvin 的每个质量维度添加两个额外的问题。
- 19.5 McCall 质量因素是在 20 世纪 70 年代提出的，自提出以来，几乎计算的每个方面都发生了翻天覆地的变化，然而，McCall 质量因素继续适用于现代软件。基于这一事实你是否能得出一些结论？
- 19.6 使用 19.2.3 节所述 ISO9126 质量因素中“维护性”的子属性，提出一组问题，探讨是否存在这些属性，仿效 19.2.4 节所示的例子。
- 19.7 用你自己的话描述软件质量困境。
- 19.8 什么是“足够好”的软件？说出具体公司的名字，以及你认为运用足够好思想开发的具体产品。
- 19.9 考虑质量成本的 4 个方面，你认为哪个方面是最昂贵的，为什么？
- 19.10 进行网络搜索，寻找直接由低劣的软件质量所致“风险”的其他三个例子。考虑从 <http://catless.ncl.ac.uk/risks> 开始搜索。
- 19.11 质量和安全是一回事吗？请加以解释。
- 19.12 解释为什么我们许多人仍在沿用 Meskimen 定律，使得软件业如此的原因是什么？

扩展阅读与信息资源

Chemutri (《Master Software Quality Assurance : Best Practices, Tools and Techniques for Software Developers》, Ross Publishing, 2010)、Henry 和 Hanlon (《Software Quality Assurance》, Prentice Hall, 2008)、Khan 和他的同事 (《Software Quality : Concepts and Practice》, Alpha Science

International, Ltd., 2006)、O'Regan (《A Practical Approach to Software Quality》, Springer, 2002) 以及 Daughtrey (《Fundamental Concepts for the Software Quality Engineer》, ASQ Quality Press, 2001) 的书讨论了软件质量的基本概念。

Duvall 和他的同事 (《Continuous Integration: Improving Software Quality and Reducing Risk》, Addison-Wesley, 2007)、Tian (《Software Quality Engineering》, Wiley-IEEE Computer Society Press, 2005)、Kandt (《Software Engineering Quality Practices》, Auerbach, 2005)、Godbole (《Software Quality Assurance: Principles and Practice》, Alpha Science International, Ltd., 2004) 以及 Galin (《Software Quality Assurance: From Theory to Implementation》, Addison-Wesley, 2003) 介绍了软件质量保证的具体做法。Sterling (《Managing Software Debt》, Addison-Wesley, 2010) 以及 Stamelos 和 Sfetos (《Agile Software Development Quality Assurance》, IGI Global, 2007) 讨论了在敏捷过程环境中的质量保证问题。

可靠的设计可以实现软件的高质量。Jayasawal 和 Patton (《Design for Trustworthy Software》, Prentice Hall, 2006) 以及 Ploesch (《Contracts, Scenarios and Prototypes》, Springer, 2004) 讨论了开发“健壮”软件的工具和技术。

测量是软件质量工程的一个重要部分。Jones 和 Bonsignour (《The Economic of Software Quality》, Addison-Wesley, 2011)、Ejiogu (《Software Metrics: The Discipline of Software Quality》, BookSurge Publishing, 2005)、Kan (《Metrics and Models in Software Quality Engineering》, Addison-Wesley, 2002) 以及 Nance 和 Arthur (《Managing Software Quality》, Springer, 2002) 讨论了重要的质量相关的测度和模型。Evans (《Achieving Software Quality through Teamwork》, Artech House Publishers, 2004) 考虑了软件质量中团队方面的问题。

软件质量方面的大量信息可以在网上获得, 关于软件质量方面最新的参考文献可以在 SEPA 网站 www.mhhe.com/pressman 的“software engineering resources”下找到。

评审技术

要点浏览

概念: 在开发软件工作产品时可能会犯错误，这并不是羞耻的事，只要在产品交付最终用户之前，努力、很努力地发现并纠正错误即可。技术评审是在软件过程早期最有效的查错机制。

人员: 软件工程师和同事一起进行技术评审，也叫同行评审。

重要性: 如果在软件过程的早期发现错误，修改的成本就较少。另外，随着软件过程的推进，错误会随之放大，因此，过程早期留下的没有处理的小错误，可能在项目后期放大成一组严重的错误。最后，通过减少项目后期所需的返工，评

审为项目节省了时间。

步骤: 评审方法因所选评审的正式程度而异，对不同类型的评审，不是所有步骤都用到，一般分 6 个步骤：计划、准备、组织会议、记录错误、进行修改（评审之后做）、验证是否恰当地进行了修改。

工作产品: 评审的输出是发现问题和错误的清单。另外，还标示出工作产品的技术状态。

质量保证措施: 首先，选择适合开发文化的评审类型，然后，遵守保证评审成功的指导原则，如果进行的评审有利于得到高质量的软件，说明你做对了。

软件评审是软件过程中的“过滤器”。也就是说，在软件工程过程的不同阶段进行软件评审，可以起到发现错误和缺陷，进而消除它们的作用。软件评审还能够“净化”需求模型、设计模型、源代码和测试数据等软件工作产品。对于评审的需要，Freedman 和 Weinberg [Fre90] 是这样论述的：

技术工作需要评审正像铅笔需要橡皮：人非圣贤，孰能无过。我们需要技术评审的第二个理由是：尽管人们善于发现自己的某些错误，但是许多情况下犯错误的人发现自己错误的的能力远小于其他人。因此，评审过程是对 Robert Burns 的祈祷的解答：

哦，聚集才能给我们力量

像别人看待我们那样，看待自己

评审（任何评审）是使用人群之间的差异达到以下目的：

1. 指出个人或团队的产品中需要改进的地方。
2. 确认产品中不期望或不需要改进的部分。

3. 与没有评审相比，得到质量更统一或至少更可预测的技术工作，以使技术工作更加可管理。

关键概念

隐错
成本效益
缺陷放大
缺陷
错误密度
错误
非正式评审
记录保存
评审报告
样本驱动评审
技术评审

在软件工程过程中可以进行的评审有很多种，它们各有各的作用。在休息室里讨论技术问题的非正式会谈就是一种评审方式。将软件架构正式介绍给客户、管理层和技术人员也是一种评审方式。但是，本书将重点讨论技术评审，即同行评审，通过举例说明非正式评审、走查和审查。从质量控制的角度出发，技术评审（Technical Review, TR）是最有效的过滤器。由软件工程师（以及其他人员）对软件工程师进行的技术评审是一种发现错误和提高软件质量的有效手段。

建议 评审就像软件过程工作流程中的过滤器一样，评审太少， workflow 就是“脏”的。评审太多， workflow 则会很慢。使用度量以确定哪些评审起作用，并进行加强。从 workflow 中去除无效的评审以加速软件过程。

20.1 软件缺陷对成本的影响

在软件过程的环境中，术语缺陷和故障是同义词，两者都是指在软件发布给最终用户（或软件过程内其他框架活动）后发现的质量问题。在前面几章，我们使用术语错误来描绘在软件发布给最终用户（或软件过程内其他框架活动）之前软件工程师（或其他人）发现的质量问题。

信息栏 隐错、错误和缺陷

软件质量控制的目标是消除软件中存在的^①质量问题，从广义上讲，这也是一般质量管理的目标。有很多术语可用来描述这些质量问题，如“隐错”（bug）、“故障”（fault）、“错误”（error）或“缺陷”（defect）。它们的含义是相同的吗？还是存在微小的差别呢？

在本书中，我们明确指出了“错误”（指在软件交付给最终用户之前发现的质量问题）和“缺陷”（指在软件交付给最终用户之后发现的质量问题）的差别^②。这样区别的原因是“错误”和“缺陷”对经济、商业、心理和人员的影响有很大区别。作为软件工程师，我们期望在客户和最终用户遇到问题之前尽可能多地发现并改正“错误”。我们同样期望避免“缺陷”，因为“缺陷”

（有理由）使软件工作者显得水平低下。

然而要指出的是，本书中描述的“错误”和“缺陷”的差别并不是主流观点。在软件工程领域中，大多数人都认为“缺陷”“错误”“故障”和“隐错”是没有差别的。也就是说，遇到问题的时间与采用哪个术语描述毫无关系。这个观点中争论的焦点是：有些时候很难明确地区分时间点位于交付之前还是交付之后（如敏捷开发中采用的增量过程）。

不管怎么说都应该认识到：发现问题的时间点是^③非常关键的。软件工程师应该努力再努力，力图在他们的客户和最终用户遇到问题之前将其发现。有关“隐错”术语的讨论，感兴趣的读者可从 www.softwaredevelopment.ca/bugs.shtml 找到。

正式技术评审的主要目标是在软件过程中发现错误，以使它们不会在软件交付之后变成缺陷。正式技术评审最明显的优点就是可以早些发现错误，以防止将错误传递到软件过程的后续阶段。

产业界的大量研究表明：设计活动引入的错误占软件过程中出现的所有错误（和最终

① 如果考虑的是软件过程改进，则在过程框架活动之间（如从建模到构造）传递的质量问题同样可以叫作“缺陷”，因为在一个工作产品（如设计模型）“交付”给下一个活动之前就应该发现这个问题。

的所有缺陷)数量的 50%~65%。然而,已经证明,评审技术在发现设计缺陷方面有高达 75% 的有效率 [Jon86]。通过检测和消除大量设计错误,评审过程将极大降低软件过程后续活动的成本。

20.2 缺陷的放大和消除

可以用“缺陷放大模型”[IBM81]来说明在软件工程过程的设计和编码活动中错误的产生和检测。该模型如图 20-1 所示,其中方框表示软件工程活动。在该活动中,可能由于疏忽产生错误,评审可能没有发现新产生的错误以及来自前面步骤的错误,从而导致一定数量的错误通过了当前步骤。在某些情况下,从前面步骤传过来的错误在当前步骤中会被放大(放大倍数为 x)。将开发步骤方框进一步细分可以说明这些特点及错误检测的有效性百分比,错误检测的有效性百分比是评审完善性的函数。

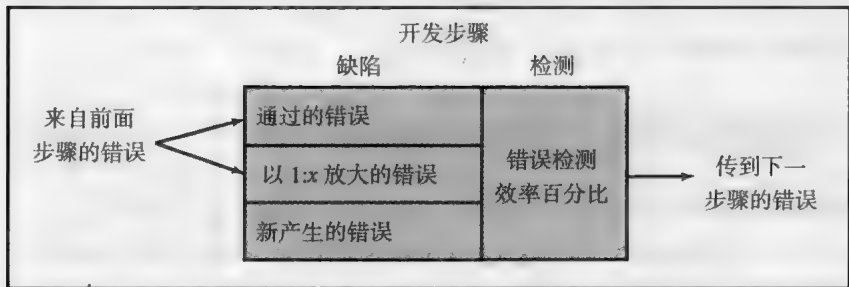


图 20-1 缺陷放大模型

图 20-2 是一个假设在软件过程中不进行任何评审的缺陷放大的例子。如图中所示,假设每个测试步骤都能够发现和改正 50% 的输入错误,而且不引入任何新的错误(乐观估计)。初步设计阶段的 10 个错误在测试开始之前就能放大为 94 个错误,12 个潜伏的错误则随软件发布进入客户现场。图 20-3 的情况与图 20-2 类似,只是在设计和编码开发步骤中引入了评审。在这种情况下,最初的 10 个初步设计错误在测试开始之前放大为 24 个错误,最后只剩 3 个潜伏的错误。回忆一下与发现和改正错误相关的相对成本,将图 20-2 和图 20-3 每个步骤中发现的错误数量乘以消除一个错误所需要的成本(设计是 1.5 个成本单位,测试前是 6.5 个成本单位,测试中是 15 个成本单位,发布后是 67 个成本单位),由此可以确定总开发成本(对我们假设的例子而言是有或没有评审情况下的成本)[⊖]。通过这些数据,在进行了评审的情况下,开发和维护的总成本是 783 个成本单位,而在不进行评审的情况下,总成本是 2177 个成本单位——几乎是前者的 3 倍。

为了进行评审,软件工程师必须花费时间和精力,开发组织也必须提供相应费用。然而,上述例子的结果已经证明了我们面临的选择:要么现在付出,否则以后会付出更多。

建议 正式技术评审的主要目标是在错误传递到另一个软件 Engineering 活动或发布给最终用户之前发现它。

引述 正如医生所说,某些疾病在其初发阶段易于治愈,但是难于识别……然而,如果它们没有在早期被发现和治疗,就将随着时间的推移变得易于识别而难于治愈。

Niccolo
Machiavelli

⊖ 这些乘数与图 19-2 给出的数据稍有不同,图 19-2 更新。但是,它们都很好地例证了缺陷放大成本。

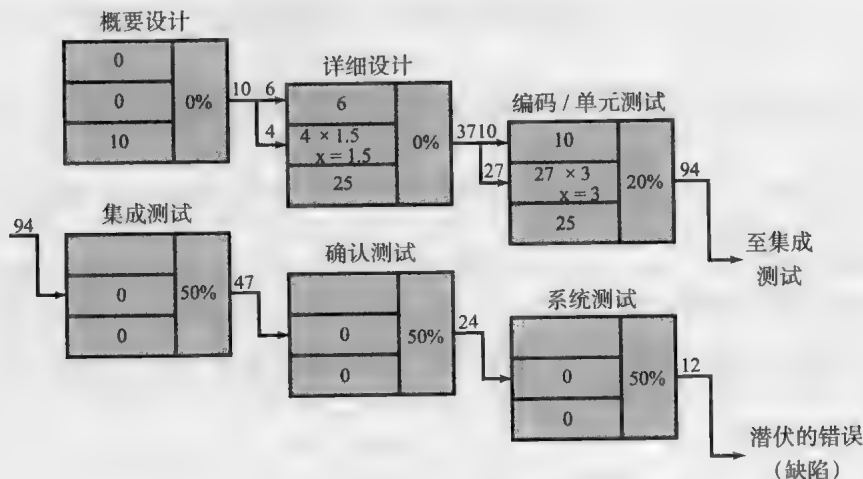


图 20-2 缺陷放大——无评审

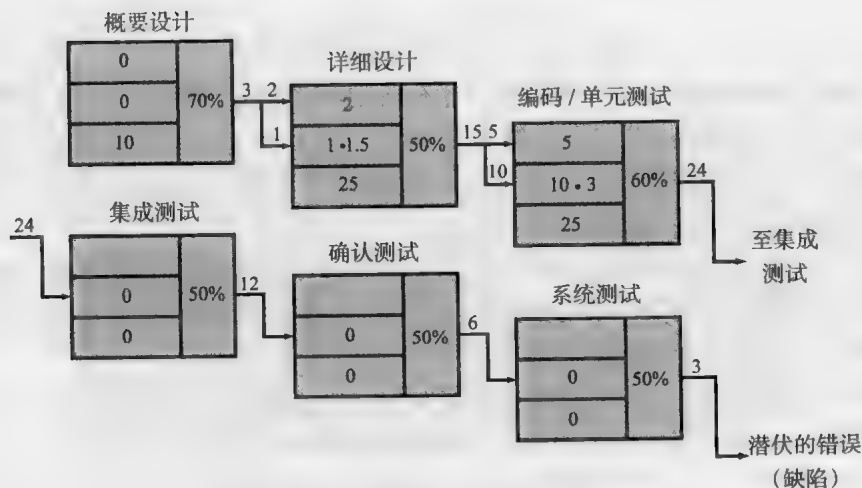


图 20-3 缺陷放大——有评审

20.3 评审度量及其应用

技术评审是好的软件工程实践所需要的很多活动之一。每项活动都需要付出人力，由于可用的项目工作量是有限的，因此重要的是，软件工程组织要定义一套可以用来评估其工作效率的度量（第30章），从而理解每项活动的有效性。

尽管可以为技术评审定义很多度量，但一个相对较小的子集就可以提供有益的见解。可以为所进行的每项评审收集以下评审度量数据：

- 准备工作量 E_p ——在实际评审会议之前评审一个工作产品所需的工作量（单位：人时）。
- 评估工作量 E_a ——实际评审工作中所花费的工作量（单位：人时）。
- 返工工作量 E_r ——修改评审期间发现的错误所用的工作量（单位：人时）。
- 工作产品规模 WPS——被评审的工作产品规模的衡量（例如，UML 模型的数量、文档的页数或代码行数）。

- 发现的次要错误 Err_{minor} ——发现的可以归为次要错误的数量（要求少于预定的改错工作量）。
- 发现的主要错误 Err_{major} ——发现的可以归为主要错误的数量（要求多于预定的改错工作量）。

通过将所评审的工作产品类型与所收集的度量数据相关联，这些度量数据可以进一步细化。

20.3.1 分析度量数据

在开始分析之前，必须进行一些简单的计算。总评审工作量 E_{review} 和发现的错误总数 Err_{tot} 定义为：

$$E_{review} = E_p + E_a + E_r$$

$$Err_{tot} = Err_{minor} + Err_{major}$$

错误密度表示评审的每单位工作产品发现的错误数。

$$\text{错误密度} = \frac{Err_{tot}}{WPS}$$

435

例如，如果评审需求模型以发现错误、不一致之处和遗漏，将有可能以一些不同的方式计算错误密度。需求模型的描述性材料共有 32 页，其中包含 18 个 UML 图。评审发现 18 处次要错误和 4 处主要错误。因此， $Err_{tot} = 22$ 。错误密度为每个 UML 图 1.2 个错误，或者说，每页需求模型有 0.68 个错误。

如果是对一些不同类型的工作产品（例如，需求模型、设计模型、代码、测试用例）进行评审，则可以通过所有评审所发现的错误总数来计算每次评审发现的错误百分比。此外，也可以计算每个工作产品的错误密度。

在为多个项目收集到许多评审数据后便可利用其错误密度的平均值估计一个新的项目中将发现的错误数。例如，如果需求模型的平均错误密度是每页 0.6 个错误，一个新的需求模型为 32 页，粗略估计，你的软件团队在评审该文档时将能发现大约 19 或 20 个错误。如果你只发现了 6 个错误，说明你在开发需求模型方面的工作非常出色或者说明评审工作做得不够彻底。

在进行了测试（第 22 ~ 26 章）之后，有可能收集到另外一些错误数据，包括在测试期间发现和纠正错误所需要的工作量，以及软件的错误密度。可以将测试期间发现和纠正错误的相关成本与评审期间的成本相比较，这部分内容在 20.3.2 节讨论。

20.3.2 评审的成本效益

实时地测量任何技术评审的成本效益都是困难的。只有在评审工作已经完成，已收集了评审数据，计算了平均数据，并测量了软件的下游质量（通过测试）之后，软件工程组织才能够对评审的有效性和成本效益进行评估。

返回 20.3.1 节所介绍的例子，需求模型的平均错误密度确定为每页 0.6 个错误。修改一个次要模型错误需要 4 人时（评审后立即修改），修改一个主要需求错误需要 18 人时。对所收集的评审数据进行分析，发现次要错误出现的频度比主要错误出现的频度高 6 倍。因此，

436

可以估计，评审期间查找和纠正需求错误的平均工作量大约为 6 人时。

对于测试过程中发现的与需求有关的错误，查找和纠正的平均工作量为 45 人时（对于

错误的相对严重性没有任何数据可用)。使用提到的平均数,我们得到:

$$\text{每个错误节省的工作量} = E_{\text{testing}} - E_{\text{reviews}} = 45 - 6 = 39 \text{ 人时 / 错误}$$

由于在需求模型评审时发现了 22 个错误,因此节省了约 858 人时的测试工作量。而这只是与需求有关的错误,与设计和代码相关的错误将加入到整体效益中。无疑,节省的工作量使交付周期缩短,上市时间提前了。

Karl Wiegars [Wie02] 在他的有关同行评审的书讨论了从大公司得到的传闻数据,这些大公司已经使用审查(一种比较正式的技术评审)作为软件质量控制活动的一部分。Hewlett Packard 称审查有 10 : 1 的投资回报率,并指出实际产品交付时间平均提前了 1.8 个月。AT & T 公司表示,审查使软件错误总成本降低到原来的 1/10,质量提高了一个数量级,而且生产率提高了 14%。还有其他类似的效益报告。技术评审(为设计和其他技术活动)提供了明显的成本效益,并且确实节省了时间。

但对于许多软件开发人员,这种说法违反直觉。软件开发人员认为“评审需要时间,我们没有多余的时间!”他们认为,在每个软件项目中时间是一种宝贵的商品,评审“每个工作产品细节”占用了太多时间。

本节先前提出的例子显示不是这样。更重要的是,对软件评审行业数据的收集已有 20 多年,可以用图形的方式定性地进行概括,如图 20-4 所示。

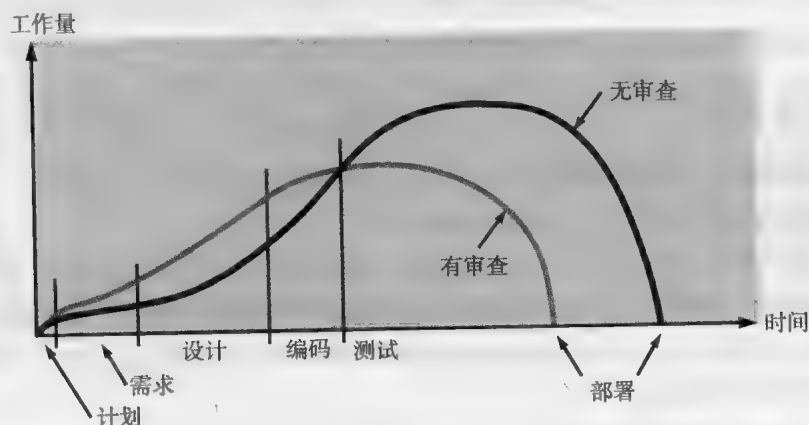


图 20-4 有评审和没有评审时花费的工作量 [Fag86]

从图 20-4 可以看到,使用评审时花费的工作量在软件开发的早期确实是增加的,但是评审的早期投入产生了效益,因为测试和修改的工作量减少了。重要的是,有评审开发的发布日期比没有评审时要快。评审不费时间,而是节省时间。

437

20.4 评审的正式程度

应该以某种正式程度应用技术评审,该正式程度应该适合所生产的产品、项目的时间线和做评审工作的人。图 20-5 描述了技术评审的参考模型 [Lai02],该模型中的 4 个特征有助于决定进行评审的形式。

参考模型的每个特征都有助于确定评审的正式程度。在下述条件下评审的正式度会提高:(1)明确界定每位评审人员的不同职责;(2)为评审进行充分的计划和准备;(3)为评审定义清晰的结构(包括任务和内部工作产品);(4)评审人员对所做修改的后续跟踪。

为理解参考模型,让我们假设你已决定评审 SafeHomeAssured.com 的界面设计。你可

438

以以各种不同的方式进行评审，从相对非正式的到极其严格的。如果你觉得非正式的方法更适合，那么你可以要求一些同事（同行）检查界面原型，努力发现潜在的问题。大家认为不需要进行事先准备，但也可以以合理的结构化方式来评估原型——首先查看布局，接下来看是否符合美学，再接下来是导航选项，等等。作为设计者，你可以记些笔记，但不用那么正式。

但是如果界面是整个项目成败的关键呢？如果人的生命依赖于完全符合人体工程学的界面呢？这时你可能会认为更严格的做法是必要的。于是成立一个评审小组，小组中的每个人承担特定的职责，如领导小组工作、记录发现的问题、提供材料，等等。评审之前每个评审人员将有机会接触工作产品（该例中即为界面原型），花时间查找错误、不一致和疏漏。按照评审之前制定的议程执行一组任务。正式记录评审的结果，评审小组根据评审结果对工作产品的状态做出决议。评审小组的成员可能还要核实是否适当地进行了修改。

本书关注两类广为采用的技术评审：非正式评审和更为正式的技术评审。在每一类中都有一些不同的方法可以选择。这些都在接下来的几节中介绍。

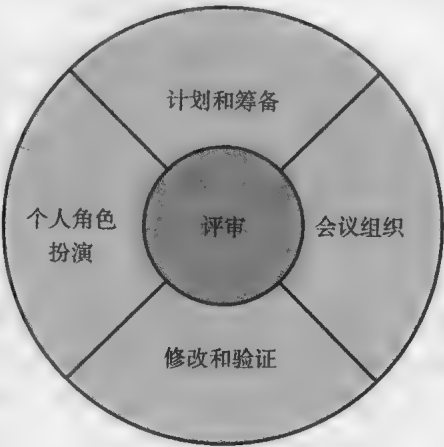


图 20-5 技术评审参考模型

20.5 非正式评审

非正式评审包括：与同事就软件工程产品进行的简单桌面检查，以评审一个工作产品为目的的临时会议（涉及两人以上），或结对编程评审（第 5 章）。

与同事进行的简单桌面检查或临时会议是一种评审。但是，因为没有事先规划或筹备工作，没有会议的议程或组织，没有对发现的错误进行后续的跟踪处理，所以这种评审的有效性大大低于更为正式的方法。但是，简单桌面检查可以也的确能发现错误，否则这些错误可能传播到后续的软件过程。

提高桌面检查评审效能的一种方法是为软件团队的每个主要的工作产品制定一组简单评审检查单。检查单中提出的问题是常见问题，但有助于指导评审人员检查工作产品。例如，让我们重新就 SafeHomeAssured.com 的界面原型进行桌面检查。设计师和同事使用界面的检查清单来检查原型，而不是在设计者的工作站上简单地操作原型：

- 布局设计是否使用了标准惯例？是从左到右还是从上到下？
- 显示是否需要滚动？
- 是否有效使用了不同的颜色和位置以及字体和大小？
- 所有导航选项或表示的功能是否在同一抽象级别？
- 所有导航选择是否清楚地标明了？

.....

评审人员指出的任何错误和问题由设计人员记录下来以在稍后的时间进行解决。桌面检查可以以一个特别的方式安排，或者也可以授权作为良好的软件工程实践的一部分。一般来说，桌面检查评审材料的数量相对较少，总体上花费时间大致在一两个小时。

第 5 章描述了结对编程，方式如下：极限编程建议，两个人一起在一台计算机工作站编

439

写一段代码。这提供了一种机制，可以实时解决问题（两人的智慧胜过一人）并获得实时的质量保证。

结对编程的特点是持续的桌面检查。结对编程鼓励在创建工作产品（设计或代码）时进行持续的审查，而不是在某个时候安排评审。好处是即时发现错误，结果是得到更好的工作产品质量。

在讨论结对编程的效能时，Williams 和 Kessler[Wil00] 这样描述：

轶事和初步统计证据表明，结对编程是一种强大的技术，可以有效创造高品质的软件产品。两人一起工作并分享想法以共同解决复杂的软件开发问题。他们不断检查彼此的产品，从而以最有效的形式尽早去除缺陷。此外，他们彼此一心一意专注于手头的任务。

一些软件工程师认为，结对编程固有的冗余是浪费资源。毕竟，为什么为两个人指派一个人可以完成的工作？对这个问题的回答可以在 20.3.2 节找到。如果作为结对编程的结果生产出的工作产品的质量明显优于单个人的工作，那么在质量方面的节约足以弥补结对编程带来的“冗余”。

信息栏 评审检查单

即使评审工作已经很好地组织并严格地进行，为评审人员提供一张“参考清单”也是一个不错的主意。也就是说，需要有这样一张检查单，它为每位评审人员对正在进行评审的具体的工作产品列出要问的问题。

最全面的评审检查单集合之一是美国航天局在戈达德空间飞行中心（Goddard

Space Flight Center）开发的，可以在 <http://www.hq.nasa.gov/office/codeq/software/Complex-Electronics/checklists.htm> 得到。

其他有用的技术评审检查单有：

- Process Impact。 www.processimpact.com/pr_goodies.shtml。
- Macadamian。 www.macadamian.com。

440

20.6 正式技术评审

正式技术评审（FTR）是一种由软件工程师（以及其他人员）进行的软件质量控制活动。FTR 的目标是：（1）发现软件的任何一种表示形式中的功能、逻辑或实现上的错误；（2）验证评审中的软件是否满足其需求；（3）保证软件的表示符合预先指定的标准；（4）获得以统一的方式开发的软件；（5）使项目更易于管理。除此之外，FTR 还提供了培训机会，使初级工程师能够了解软件分析、设计和实现的不同方法。由于 FTR 的进行，使大量人员对软件系统中原本并不熟悉的部分更为了解，因此，FTR 还起到了培训后备人员和促进项目连续性的作用。

FTR 实际上是一类评审方式，包括走查（walkthrough）和审查（inspection）。每次 FTR 都是以会议形式进行的，只有经过适当的计划、控制和参与，FTR 才能获得成功。在后面的几节中，我们将给出类似于走查的典型正式技术评审指导原则。如果你对审查以及走查的其他信息感兴趣，可参见 [Rad02]、[Wie02] 或 [Fre90]。

引述 没有任何事情能比一个人去校对另一个人的工作更令人振奋了。

Mark Twain

20.6.1 评审会议

不论选用何种 FTR 方式，每个评审会议都应该遵守以下约束：

- 评审会（通常）应该由 3～5 人参加。
- 应该提前进行准备，但是占用每人的工作时间应该不超过 2 小时。
- 评审会的时间应该少于 2 小时。

考虑到这些限制，显然 FTR 应该关注的是整个软件中的某个特定（且较小的）部分。比如说，只走查各个构件或者一小部分构件，不要试图评审整个设计。FTR 关注的范围越小，发现错误的可能性越大。

FTR 关注的是某个工作产品（例如，一部分需求模型、一份详细的构件设计、一个构件的源代码）。开发这个工作产品的个人（生产者）通知项目负责人“该工作产品已经完成，需要进行评审”。项目负责人与评审主席取得联系，由评审主席负责评估该工作产品是否准备就绪，制作产品材料副本，并将这些副本分发给 2～3 位评审员以便事先做准备。每位评审员应该用 1～2 个小时来评审该工作产品，通过做笔记或者其他方法熟悉该工作产品。与此同时，评审会主席也应该评审该工作产品，并制定评审会议的日程表，通常会安排在第二天开会。

评审会议由评审会主席、所有评审员和开发人员参加。其中一位评审员还充当记录员的角色，负责记录（书面的）在评审过程中发现的所有重要问题。FTR 一般从介绍会议日程并由开发人员做简单的介绍开始。然后由开发人员“走查”该工作产品，并对材料做出解释，而评审员则根据预先的准备提出问题。当发现了明确的问题或错误时，记录员逐一加以记录。

在评审结束时，所有 FTR 与会者必须做出以下决定中的一个：（1）可以不经修改而接受该工作产品；（2）由于严重错误而否决该工作产品（错误改正后必须再次进行评审）；（3）暂时接受该工作产品（发现了一些必须改正的小错误，但是不再需要进行评审）。做出决定之后，所有 FTR 与会者都需要签名，以表示他们参加了此次 FTR，并且同意评审小组所做的决定。

20.6.2 评审报告和记录保存

在 FTR 期间，由一名评审员（记录员）主动记录所有提出的问题。在评审会议结束时要对此类问题进行汇总，并生成一份“评审问题清单”。此外，还要完成一份“正式技术评审总结报告”。评审总结报告中要回答以下 3 个问题：

1. 评审的产品是什么？
2. 谁参与了评审？
3. 发现的问题和结论是什么？

评审总结报告通常只是一页纸的形式（可能还有附件）。它是项目历史记录的一部分，有可能将其分发给项目负责人和其他感兴趣的参与方。

评审问题清单有两个作用：（1）标识产品中存在问题的区域；（2）作为行动条目检查单以指导开发人员进行改正。通常将评审问题清单附在总结报告的后面。

为了保证评审问题清单中的每一条都得到适当的改正，建立跟踪规程非常重要。只有做

网络资源 《NASASATC 正式评审指南》可从 http://www.everyspace.com/NASA/NASASATC-Gen-eral/NASAGB-A302_2418 下载。

关键点 FTR 关注的是某工作产品中比较小的一部分。

建议 在某些情况下，让其他人而不是开发人员本人走查被评审的产品是一个很好的方法，这样可得到对工作产品的真实解释且更易于发现错误。

到这一点,才能保证提出的问题真正得到“解决”。方法之一就是将跟踪的责任指派给评审会主席。

20.6.3 评审指导原则

进行正式技术评审之前必须制定评审的指导原则,并分发给所有评审员以得到大家的认可,然后才能依照它进行评审。不受控制的评审通常比没有评审还要糟糕。下面列出了最低限度的一组正式技术评审指导原则:

1. 评审工作产品,而不是评审开发人员。FTR 涉及他人和自己。如果进行得适当,FTR 可以使所有参与者体会到温暖的成就感。如果进行得不适当,则可能陷入一种审问的气氛之中。应该温和地指出错误,会议的气氛应该是轻松的和建设性的,不要试图贬低或羞辱他人。评审会主席应该引导评审会议,确保维护适当的气氛和态度,应立即终止已变得失控的评审。
 2. 制定并遵守日程表。各种类型会议的主要缺点之一就是放任自流。必须保证 FTR 不要离题且按照计划进行。评审主席负有维持会议程序的责任,在有人转移话题时应该提醒他。
 3. 限制争论和辩驳。当评审员提出问题时,未必所有人都认同该问题的严重性。不要花时间去争论这类问题,这类问题应该被记录在案,留到会后进行讨论。
 4. 要阐明问题,但是不要试图解决所有记录的问题。评审不是一个解决问题的会议,问题的解决应该由开发人员自己或是在个别人帮助下放到评审会议之后进行。
 5. 做笔记。有时候让记录员在黑板上做笔记是一种很好的方式,这样,在记录员记录信息时,其他评审员可以推敲措辞,并确定问题的优先次序。或者,笔记可直接输入到笔记本电脑中。
 6. 限制参与者人数,并坚持事先做准备。虽然两个人比一个人好,但 14 个人并不一定就比 4 个人好。应该根据需要将参与评审的人员数量限制到最低,而且所有参与评审的小组成员都必须事先做好准备。评审会主席应该向评审员要求书面意见(以表明评审员的确对材料进行了评审)。
 7. 为每个将要评审的工作产品建立检查单。检查单能够帮助评审会主席组织 FTR 会议,并帮助每位评审员将注意力集中到重要问题上。应该为分析、设计、代码甚至测试等工作产品建立检查单。
 8. 为 FTR 分配资源和时间。为了进行有效的评审,应该将评审作为软件过程中的任务列入进度计划,而且还要为由评审结果所引发的不可避免的修改活动分配时间。
 9. 对所有评审员进行有意义的培训。为了提高效率,所有评审参与者都应该接受某种正式培训。培训要强调的不仅有与过程相关的问题,而且还应该涉及评审的心理学方面。Freedman 和 Weinberg [Fre90] 估计每 20 人进行为期一个月的学习,将能够使其更有效地参与评审。
 10. 评审以前所做的评审。听取汇报对发现评审过程本身的问题十分有益,最早被评审的工作产品本身就是评审的指导原则。
- 由于成功的评审涉及许多变数(如参与者数量、工作产品类型、时间

建议 不要严厉地指出错误。可以采用较温和的方式,比如问一个问题,使开发人员能够自己发现错误。

引述 经常是几分钟的会议,却浪费了几个小时。
作者不详

引述 生活中最美好的回报之一就是:凡是努力真诚帮助别人的人,同时也是在帮助自己。

Ralph Waldo
Emerson

442

443

和长度、特定的评审方法等), 软件组织应该在实验中确定何种方法对自己的特定环境最为适用。

20.6.4 样本驱动评审

在理想情况下, 每个软件工作产品都要经过正式技术评审。但在实际的软件项目中, 由于资源有限和时间不足, 即使意识到评审是一种质量控制的机制, 评审也常常被省略。

Thelin 和他的同事 [The01] 提出了样本驱动评审过程, 在这个过程中, 要对所有软件工作产品的样本进行审查, 以决定哪些工作产品是最有错误倾向的, 然后集中全部 FTR 资源, (根据抽样过程中收集的数据) 只分配给那些可能具有错误倾向的工作产品。

为了提高效率, 样本驱动评审过程必须对作为整个 FTR 的主要目标的那些工作产品进行量化。量化时一般采用以下步骤 [The01]:

1. 审查每个软件工作产品 i 的若干分之一, 记做 $1/a_i$, 记录在其中发现的缺陷数量 f_i 。
2. 用 f_i 除以 a_i 可得到在工作产品 i 中缺陷数量的粗略估算值。
3. 按照缺陷数量粗略估算值的递减次序排列这些工作产品。
4. 将现有的评审资源集中到那些具有最高缺陷数量估算值的工作产品上。

建议 评审要花费时间, 但很值得。然而, 如果时间紧迫而你又别无选择, 也不要省略评审, 最好采用样本驱动评审。

从工作产品中抽样的这一小部分必须能代表整个工作产品, 并且要足够大, 对进行抽样的评审者来说有意义。当 a_i 增大时, 样本有效代表工作产品的可能性也随之增长, 而进行抽样所需要的资源也随之增长。开发各种类型的工作产品时, 软件工程团队必须确定 a_i 的最佳取值。^①

444

SafeHome

质量问题

[场景] Doug Miller 的办公室, SafeHome 软件项目刚开始的时候。

[人物] Doug Miller (SafeHome 软件团队经理) 以及软件工程团队的其他成员。

[对话]

Doug: 我知道, 过去我们没有花时间去为这个项目建立质量计划。但是现在项目已经启动, 我们必须考虑质量……对吗?

Jamie: 是的。我们已经决定了, 在建立需求模型 (第 9 ~ 11 章) 的时候, Ed 答应负责为每个需求建立测试规程。

Doug: 那太好了, 可是我们不会等到测试

的时候才来评估质量吧, 是不是?

Vinod: 不会! 当然不会。我们已经将评审的进度安排纳入到这个软件增量的项目计划之中了。我们将通过评审开始质量控制。

Jamie: 我有点担心我们没有足够的时间来进行所有的评审。事实上, 我也知道会这样。

Doug: 嗯。那你觉得该怎么办呢?

Jamie: 我认为我们应该选择分析和设计模型中对 SafeHome 最关键的元素进行评审。

Vinod: 可是如果我们遗漏了模型中某个

^① Thelin 和他的同事已经进行了详细的模拟, 这些有助于确定 a_i 的取值, 详见 [The01]。

部分而没有评审，怎么办？

Shakira：我阅读了有关抽样技术方面的资料（20.6.4节），可以帮助我们明确应评审的元素。（Shakira描述了这种方法。）

Jamie：也许……但是，我也不能肯定我们是否有时间对模型中的每个元素进行抽样。

Vinod：Doug，你想让我们怎么做？

Doug：参照极限编程（第5章）。我们结对

（两个人）找出每个模型中的元素，并同时
进行非正式的评审。之后我们将能够明确
“关键”元素，以进行更正式的团队评审，
但是应将这样的评审减到最少。这样的话，
所有的事情都不只一组人员检查过了，但
我们仍然可以维持原来的交付日期。

Jamie：就是说我们必须重新调整进度。

Doug：就是这样。在这个项目上质量高于
进度。

20.7 产品完成后评估

如果软件团队在将软件交付给最终用户后花些时间评估软件项目的结果，那么他们会学到很多东西。在一个特定项目中应用软件工程过程和实践时，Baaz和他的同事[Baa10]建议使用产品完成后评估（Post-Mortem Evaluation, PME）作为一种机制来确定什么是对的、什么是错的。

不同于FTR专注于特定的工作产品，PME检查整个软件项目，重点放在“优点（成绩和正面的经验）和挑战（问题和负面的体验）”[Baa10]。PME通常以研讨的形式进行，受到了软件团队成员和利益相关者的关注，其目的是识别优点和挑战，从这两个方面吸取经验和教训；目标是提出过程和促进实践向前发展的改进意见。

445

20.8 小结

每次技术评审的目的都是找出错误和发现可能对将要部署的软件产生负面影响的问题。越早发现并纠正错误，错误传播到其他软件工程产品并扩大的可能性就越小，继而导致需要更多的工作来纠正错误的可能性也越小。

为了确定质量控制活动是否在起作用，应该收集一组度量。评审度量的重点放在进行评审需要的工作量、评审中发现的错误的类型和严重程度方面。一旦收集了度量数据，就可以用来评估你所进行的评审的效率。行业数据显示评审能够提供可观的投资回报。

评审正式程度的参考模型以评审人员承担的职责、计划和筹备、会议结构、纠正的办法和验证为特征，这些特征表明了进行评审的正式程度。非正式评审在性质上是临时的，但仍然可以有效地用于发现错误。正式评审更有条理，最有可能产生高质量软件。

非正式评审的特点是最低限度的规划和准备，并少有记录。桌面检查和结对编程属于非正式评审。

正式技术评审是一个程式化的会议，已证明在发现错误方面是非常有效的。走查和审查为每个评审人员建立了确定的职责，鼓励计划和事先准备，需要应用规定的评审原则、授权记录和状态报告。当不可能对所有工作产品进行正式技术评审时，可以使用样本驱动评审。

习题与思考题

20.1 解释错误和缺陷的不同。

- 20.2 为什么我们不能只是等到测试的时候才去发现和纠正所有的软件错误？
- 20.3 假设需求模型引入了 10 个错误，每个错误按 2 : 1 的比例在设计阶段放大，设计阶段引入了另外的 20 个错误，并且这 20 个错误按 1.5 : 1 的比例在编码阶段放大，在编码阶段又引入了另外 30 个错误。进一步假设，所有单元测试会发现所有错误的 30%，集成测试将找到剩余错误的 30%，验证测试会发现剩余错误的 50%。若没有进行评审，则有多少错误将被发布出去？
- 20.4 重新考虑习题 20.3 所述情形，但现在假设进行了需求评审、设计评审和代码评审，并且这些步骤能有效地发现所有错误的 60%。那么有多少错误将被发布出去？
- 20.5 重新考虑习题 20.3 和习题 20.4 所述情形，对于每个已发布的错误，发现和改正的成本是 4800 美元，在评审时发现并改正每个错误花费 240 美元，通过进行评审可节约多少钱？
- 20.6 用你自己的话描述图 20-4 的含义。
- 20.7 你认为参考模型的哪一个特征对评审的正式程度影响最大？解释原因。
- 20.8 桌面检查可能造成问题而没有带来好处，你能想到几个实例吗？
- 20.9 仅当每个参与者都事先进行了准备时，正式技术评审才是有效的。你如何识别没有准备好的评审参与者？如果你是评审主席，你该如何做？
- 20.10 考虑 20.6.3 节提出的所有评审准则，你认为哪一条是最重要的，为什么？

扩展阅读与信息资源

在过去的 10 年中，软件评审方面的书籍相对较少。最近出版的有指导价值的书包括：McCann (《Cost-Benefit Analysis of Quality Practices》，IEEE Press, 2012)，Wong (《Modern Software Review》，IRM Press, 2006)，以及 Young (《Project Requirements: A Guide to Best Practices》，Management Concepts, 2006)。旧一些的有指导价值的书包括：Radice (《High Quality》，Low Cost Software Inspections, Paradoxicon Publishers, 2002)，Wiegers (《Peer Reveiws in Software: A Practical Guide》，Addison-Wesley, 2001)，以及 Gilb 和 Graham (《Software Inspection》，Addison-Wesley, 1993)。Freedman 和 Weinberg 的书 (《Handbook of Walkthroughs, Inspections and Technical Reviews》，Dorset House, 1990) 仍是一部经典的教材，提供了有关软件评审这个重要课题的有用信息。

Rubin (《Essential Scrum: A Practical Guide to the Most Popular Agile Process》，Addison-Wesley, 2012) 和 Adkins (《Coaching Agile Teams: A Companions for ScrumMasters, Agile Coaches, and Project Managers in Transition》，Addison-Wesley, 2010) 描述了评审在敏捷软件过程中的作用。

软件评审方面多种多样的信息都可在网上得到。软件评审相关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

软件质量保证

要点浏览

概念：仅仅嘴上说“软件质量重要”是不够的，还必须：(1) 明确给出你所说的“软件质量”的含义；(2) 提出一组活动，这些活动将有助于保证每个软件工程项目表现出高质量；(3) 对每个软件项目实施质量控制和质量保证活动；(4) 运用度量技术来制定软件过程改进的策略，进而提高最终产品的质量。

人员：软件工程过程中涉及的每个人都要对质量负责。

重要性：要么一次做好，要么重做。如果软件团队在每个软件工程活动中都强调质量，则会减少返工量，进而降低成本，更重要的是可以缩短面市时间。

步骤：在软件质量保证活动启动前，必须按照多种不同的抽象层次来定义“软件

质量”。理解了质量的含义之后，软件团队必须确定一组软件质量保证 (SQA) 活动来过滤掉工作产品中的错误，以免这些错误再继续传播下去。

工作产品：为了制定软件团队的 SQA 策略，需要建立“软件质量保证计划”。在建模、编码阶段，主要的 SQA 工作产品是技术评审的输出 (第 20 章)；在测试阶段 (第 22 ~ 26 章)，主要的 SQA 工作产品是制定的测试计划和测试规程。也可能产生其他与过程改进相关的工作产品。

质量保证措施：在错误变成缺陷之前发现它！也就是说，尽量提高缺陷排除效率 (第 30 章)，进而减少软件团队不得不付出的返工量。

本书所讨论的软件工程方法只有一个目标：在计划时间内生产出高质量的软件。但是很多读者都会遇到这样一个问题：什么是软件质量？

Philip Crosby [Cro79] 在他的有关质量的且颇具影响力的著作中间接地回答了这个问题：

质量管理的问题不在于人们不知道什么是质量，而在于人们自认为知道什么是质量……

每个人都需要质量 (当然，是在特定条件下)，每个人都觉得自己理解它 (尽管人们不愿意解释它)，每个人都认为只要顺其自然就可以 (毕竟，我们都还做得不错)。当然，大多数人都认为这一问题都是由他人引起的。(只要他们肯花时间就能把事情做好。)

确实，质量是一个具有挑战性的概念，我们已经在第 19 章进行了详细论述。^①

有些软件开发者仍然相信软件质量是在编码完成之后才应该开始担心的事情。这是完全错误的！软件质量保证 (通常称为质量管理) 是适用于整个软件过程的一种普适性活动 (第 3 章)。

① 如果你还没有阅读第 19 章，那么现在应该阅读了。

软件质量保证 (SQA) 包括: (1) SQA 过程; (2) 具体的质量保证和质量控制任务 (包括技术评审和多层次测试策略); (3) 有效的软件工程实践 (方法和工具); (4) 对所有软件工作产品及其变更的控制 (第 29 章); (5) 保证符合软件开发标准的规程 (在适用的情况下); (6) 测量和报告机制。

本章侧重于管理问题和特定的过程活动, 以使软件组织确保“在恰当的时间以正确的方式做正确的事情”。

21.1 背景问题

对于任何为他人生产产品的企业来说, 质量控制和质量保证都是必不可少的活动。在 20 世纪以前, 质量控制只由生产产品的工匠承担。随着时间推移, 大量生产技术逐渐普及, 质量控制开始由生产者之外的其他人承担。

第一个正式的质量保证和质量控制方案于 1916 年由贝尔实验室提出, 此后迅速风靡整个制造行业。在 20 世纪 40 年代, 出现了更多正式的质量控制方法, 这些方法都将测量和持续的过程改进 [Dem86] 作为质量管理的关键要素。

软件开发质量保证的历史和硬件制造质量的历史同步。在计算机发展的早期 (20 世纪 50 年代和 60 年代), 质量保证只由程序员承担。软件质量保证的标准是 20 世纪 70 年代首先在军方的软件开发合同中出现的, 此后迅速传遍整个商业界的软件开发活动 [IEE93a]。延伸前述的质量定义, 软件质量保证就是为了保证软件高质量而必需的“有计划的、系统化的行动模式” [Sch98c]。质量保证责任的范围最好可以用曾经流行的一个汽车业口号来概括: “质量是头等重要的工作。”对软件来说含义就是, 各个参与者都对软件质量负有责任——包括软件工程师、项目管理者、客户、销售人员和 SQA 小组成员。

SQA 小组充当客户在公司内部的代表。也就是说, SQA 小组成员必须从客户的角度来审查软件。软件是否充分满足第 19 章中提出的各项质量因素? 软件工程实践是否依照预先制定的标准进行? 作为 SQA 活动一部分的技术规范是否恰当地发挥了作用? SQA 小组的工作将回答上述这些问题以及其他问题, 以确保软件质量得到维持。

21.2 软件质量保证的要素

软件质量保证涵盖了广泛的内容和活动, 这些内容和活动侧重于软件质量管理, 可以归纳如下 [Hor03]。

标准。IEEE、ISO 及其他标准化组织制定了一系列广泛的软件工程标准和相关文件。标准可能是软件工程组织自愿采用的, 或者是客户或其他利益相关者责成采用的。软件质量保证的任务是要确保遵循所采用的标准, 并保证所有的工作产品符合标准。

评审和审核。技术评审是由软件工程师执行的质量控制活动 (第 20 章), 目的是发现错误。审核是一种由 SQA 人员执行的评审, 意图是确保软件工作遵循质量准则。例如, 要对评审过程进行审核, 确保以最有可能发现错误的方

关键概念

软件质量保证的要素
形式化方法
目标
ISO 9001: 2008
标准
质量管理资源
六西格玛
软件可靠性
软件安全
SQA 计划
SQA 任务
统计软件质量保证

引述 你犯了太多错了。

Yogi Berra

449

网络资源 SQA 的深度探讨 (包含一系列定义) 可在 http://www.swwqual.com/images/FoodforThought_Jan2011.pdf 得到。

式进行评审。

测试。软件测试(第 22 ~ 26 章)是一种质量控制功能,它有一个基本目标——发现错误。SQA 的任务是要确保测试计划适当和实施有效,以便最有可能实现软件测试的基本目标。

错误 / 缺陷的收集和分析。改进的唯一途径是衡量做得如何。软件质量保证人员收集并分析错误和缺陷数据,以便更好地了解错误是如何引入的,以及什么样的软件工程活动最适合消除它们。

变更管理。变更是对所有软件项目最具破坏性的一个方面。如果没有适当的管理,变更可能会导致混乱,而混乱几乎总是导致低质量。软件质量保证将确保进行足够的变更管理实践(第 29 章)。

教育。每个软件组织都想改善其软件工程实践。改善的关键因素是对软件工程师、项目经理和其他利益相关者的教育。软件质量保证组织牵头软件过程改进(第 37 章),并是教育计划的关键支持者和发起者。

供应商管理。可以从外部软件供应商获得三种类型的软件:(1)简易包装软件包(shrink-wrapped package,例如微软 Office);(2)定制外壳(tailored shell)[Hor03]——提供可以根据购买者需要进行定制的基本框架结构;(3)合同软件(contractured software)——按客户公司提供的规格说明定制设计和构建。软件质量保证组的任务是,通过建议供应商应遵循的具体的质量做法(在可能的情况下),并将质量要求作为与任何外部供应商签订合同的一部分,确保高质量的软件成果。

安全防卫。随着网络犯罪和新的关于隐私的政府法规的增加,每个软件组织应制定对策,在各个层面上保护数据,建立防火墙保护 WebApp,并确保软件在内部没有被篡改。软件质量保证确保应用适当的过程和技术来实现软件安全(第 27 章)。

安全。因为软件几乎总是人工设计系统(例如,汽车应用系统或飞机应用系统)的关键组成部分,所以潜在缺陷的影响可能是灾难性的。软件质量保证可能负责评估软件失效的影响,并负责启动那些减少风险所必需的步骤。

风险管理:尽管分析和减轻风险(第 35 章)是软件工程师考虑的事情,但是软件质量保证组应确保风险管理活动适当进行,且已经建立风险相关的应急计划。

除了以上这些问题和活动,软件质量保证还确保将质量作为主要关注对象的软件支持活动(如维护、求助热线、文件和手册)可以高质量地进行和开展。

450

引述 卓越在于提高所供产品质量的能力是无限的。

Rick Petin

451

信息栏 质量管理资源

网上有很多质量管理资源,包括专业团体、标准组织和一般的信息资源。从以下网址开始查询是不错的选择:

- 美国质量协会(ASQ)软件分会: www.asq.org/software。
- 美国计算机协会(ACM): www.acm.org。
- 网络安全和信息系统信息分析中心(CSI-AC): <https://sw.thecsiac.com/>。

- 国际标准化组织(ISO): www.iso.ch。
- ISO SPICE: <http://www.spiceusergroup.org/>。
- 美国波多里奇国家质量奖(Malcolm Baldrige National Quality Award): <http://www.nist.gov/baldrige/>。
- 软件工程研究所: www.sei.cmu.edu/。
- 软件测试和质量工程: www.stickyminds.com。

- com。
- 六西格玛资源 www.isixsigma.com/ ; www.asq.org/sixsigma/。
 - TickIT 国际 (质量认证主题): www.tic-kit.org/international.htm。

- 全面质量管理 (TQM): <http://www.isix-sigma.com/methodology/total-quality-management-tqm/> ; <http://asq.org/learn-about-quality/total-quality-management/overview/overview.html>。

21.3 软件质量保证的过程和产品特性

当我们开始讨论软件质量保证时,值得注意的是,在一个软件环境中运行良好的 SQA 步骤和方案可能在另一个软件环境中出现问题。即使在采用一致软件工程方法^①的同一个公司,不同的软件产品也可能表现出不同水平的质量 [Par11]。

解决这一困境的方法是在理解软件产品具体的质量需求之后,选取可以用来达到那些需求的过程和具体的 SQA 活动和任务。软件工程研究所的 CMMI 和 ISO 9000 标准是最常用的软件过程框架,两者各提出一套“语法和语义” [Par11],以引导软件工程实践,从而提高产品质量。通过选择框架要素以及使这些要素与某特定产品的质量要求相匹配,软件组织“协调”使用两个模型,而不是实例化一个框架的全部。

21.4 软件质量保证的任务、目标和度量

452

软件质量保证是由多种任务组成的,这些任务是两种不同的人群相联系的——这两种人群分别是做技术工作的软件工程师和负有质量计划、监督、记录、分析和报告责任的软件质量保证组。

软件工程师通过采用可靠的技术方法和措施,进行技术评审,并进行计划周密的软件测试来获得质量 (和执行质量控制活动)。

21.4.1 软件质量保证的任务

软件质量保证组的行动纲领是协助软件团队实现高品质的最终产品。软件工程研究所推荐一套质量保证活动,即质量保证计划、监督、记录、分析和报告。这些活动由独立的软件质量保证组执行 (和完成)。

编制项目质量保证计划。该计划作为项目计划的一部分,并经所有利益相关者评审。软件工程组和软件质量保证组进行的质量保证活动都受该计划支配。该计划确定要进行的评估、要进行的审核和评审、适用于项目的标准、错误报告和跟踪的规程、软件质量保证组产出的工作产品以及将提供给软件团队的反馈意见。

提问 SQA 小组的作用是什么?

参与编写项目的软件过程描述。软件团队选择完成工作的过程。软件质量保证组审查该过程描述是否符合组织方针、内部软件标准、外部要求的标准 (例如 ISO-9001), 以及是否与软件项目计划的其他部分一致。

评审软件工程活动,以验证其是否符合规定的软件过程。软件质量保证组识别、记录和跟踪偏离过程的活动,并验证是否已做出更正。

① 例如, CMMI 定义的过程和实践 (第 37 章)。

审核指定的软件工作产品以验证是否遵守作为软件过程一部分的那些规定。质量保证工作小组审查选定的产品，识别、记录并跟踪偏差，验证已经做出的更正，并定期向项目经理报告其工作成果。

确保根据文档化的规程记录和处理软件工作和工作产品中的偏差。在项目计划、过程描述、适用的标准或软件工程工作产品中可能会遇到偏差。

记录各种不符合项并报告给高层管理人员。跟踪不符合项，直到解决。

除了这些活动，软件质量保证组还协调变更的控制和管理（第 29 章），并帮助收集和分析软件度量。

引述 质量从来没有意外，它始终是崇高的意图、真诚的努力、聪明的管理和熟练的执行的結果，它表示从多个选项中所做的明智选择。

William A.
Foster

453

SafeHome 软件质量保证

[场景] Doug Miller 的办公室，SafeHome 软件项目开始。

[人物] Doug Miller（SafeHome 软件团队经理）和产品软件工程团队的其他成员。

[对话]

Doug: 非正式评审进行得怎么样了？

Jamie: 我们正在进行项目关键部分的非正式评审，这一部分在测试之前采用结对编程方法，进展比预想的要快。

Doug: 好的，可是我想让 Bridget Thorton 的 SQA 组来审查我们的工作产品，以保证项目是在遵循我们的软件过程，且是符合我们的质量目标的。

Vinod: 他们不是在做大部分的测试吗？

Doug: 是的，但 QA 要比测试做更多的事。我们需要保证文档和代码是一致的，并且保证在集成新构件时不引入错误。

Jamie: 我真的不想由于他们发现的问题而被评头论足。

Doug: 别担心，审核的重点是要检查我们的工作产品是否符合需求，以及活动的过程。我们只会使用审核结果努力改善过程和软件产品。

Vinod: 我相信这将花费更多的时间。

Doug: 从长远来看，当我们尽早发现缺陷时，这将会节省时间，如果早期发现缺陷，修复缺陷也能花费更低的成本。

Jamie: 那么，这听起来是好事。

Doug: 同样重要的是确定哪些活动引入了缺陷，将来增加评审任务来捕获这些缺陷。

Vinod: 这将帮助我们确定我们是否足够仔细地评审活动进行了抽样。

Doug: 我认为长远来看 SQA 活动将使我们成为更好的团队。

21.4.2 目标、属性和度量

执行上节所述的软件质量保证活动，是要实现一套务实的目标。

需求质量。需求模型的正确性、完整性和一致性将对所有后续工作产品的质量有很大的影响。软件质量保证必须确保软件团队严格评审需求模型，以达到高水平的质量。

设计质量。软件团队应该评估设计模型的每个元素，以确保设计模型显示出高质量，并且设计本身符合需求。SQA 寻找能反映设计质量的属性。

代码质量。源代码和相关的工作产品（例如其他说明资料）必须符合本地的编码标准，并具有易于维护的特点。SQA 应该找出那些能合理分析代码质量的属性。

质量控制有效性。软件团队应使用有限的资源，在某种程度上最有可能得到高品质的结果。SQA 分析用于评审和测试上的资源分配，评估其分配方式是否最为有效。

对于所讨论的每个目标，图 21-1[Hya96] 标出了现有的质量属性。可以使用度量数据来标明所示属性的相对强度。

目标	属性	度量
需求质量	歧义	含糊修饰词的数量（例如：许多、大量、与人友好）
	完备性	TBA 和 TBD 的数量
	可理解性	节 / 小节的数量
	易变性	每项需求变更的数量 变更所需要的时间（通过活动）
	可追溯性	不能追溯到设计 / 代码的需求数
	模型清晰性	UML 模型数 每个模型中描述文字的页数 UML 错误数
设计质量	体系结构完整性	是否存在现成的体系结构模型
	构件完备性	追溯到结构模型的构件数
	接口复杂性	过程设计的复杂性 挑选一个典型功能或内容的平均数
	模式	布局合理性 使用的模式数量
代码质量	复杂性	环路复杂性
	可维护性	设计要素（第 32 章）
	可理解性	内部注释的百分比
	可重用性	变量命名约定 可重用构件的百分比
	文档	可读性指数
质量控制效率	资源分配	每个活动花费的人员时间百分比
	完成率	实际完成时间与预算完成时间之比
	评审效率	参见评审度量（第 20 章）
	测试效率	发现的错误及关键性问题数 改正一个错误所需的工作量 错误的根源

图 21-1 软件质量的目标、属性和度量 [Hya96]

21.5 软件质量保证的形式化方法

前面几节讨论了软件质量是每个人的工作，可以通过出色的软件工程实践以及通过应用技术评审、多层次的测试策略、更好地控制软件工作产品和所做的变更、应用可接受的软件工程标准和过程框架来实现。此外，质量可定义成一组质量属性，并使用各种指标和度量进行（间接）测量。

在过去的 30 年中，软件界有一群人——虽然不多但是很坚决——提出软件质量保证应该采用一种更为形式化的方法。一段计算机程序相当于一个数学对象，每一种程序设计语言都有一套定义严格的语法和语义，而且软件需求规格说明也有严格的方法（第 28 章）。如果

454
2
455

需求模型（规格说明）和程序设计语言都以严格的方式表示，就可以采用程序正确性证明来说明程序是否严格符合其规格说明。

程序正确性证明并不是什么新的思路。Dijkstra [Dij76a] 和 Linger、Mills、Witt [Lin79] 以及其他很多人都倡导程序正确性证明，并将它与结构化程序设计概念的使用联系在一起（第 14 章）。

21.6 统计软件质量保证

统计质量保证反映了一种在产业界不断增长的趋势：质量的量化。对于软件而言，统计质量保证包含以下步骤：

1. 收集软件的错误和缺陷信息，并进行分类。
2. 追溯每个错误和缺陷形成的根本原因（例如，不符合规格说明、设计错误、违背标准、缺乏与客户的交流）。
3. 使用 Pareto 原则（80% 的缺陷可以追溯到所有可能原因中的 20%），将这 20%（重要的少数）原因分离出来。
4. 一旦找出这些重要的少数原因，就可以开始纠正引起错误和缺陷的问题。

统计质量保证这个比较简单的概念代表的是创建自适应软件过程的一个重要步骤，在这个过程中要进行修改，以改进那些引入错误的过程元素。

引述 20% 的代码含有 80% 的错误，找到错误，修正错误！

Lowell Arthur

提问 进行统计 SQA 需要哪些步骤？

21.6.1 一个普通的例子

举一个例子来说明统计方法在软件工程师中的应用。假定软件开发组织收集了为期一年的错误和缺陷信息，其中有些错误是在软件开发过程中发现的，另外一些缺陷则是在软件交付给最终用户之后发现的。尽管发现了数以百计的不同问题，但所有问题都可以追溯到下述原因中的一个（或几个）：

- 不完整或错误的规格说明（IES）。
- 与客户交流中所产生的误解（MCC）。
- 故意违背规格说明（IDS）。
- 违反程序设计标准（VPS）。
- 数据表示有错（EDR）。
- 构件接口不一致（ICI）。
- 设计逻辑的错误（EDL）。
- 不完整或错误的测试（IET）。
- 不准确或不完整的文档（IID）。
- 将设计转换为程序设计语言实现时的错误（PLT）。
- 不清晰或不一致的人机界面（HCI）。
- 其他（MIS）。

引述 适当进行的统计分析是对不确定事物的微妙解剖，是对推测的调查。

M. J. Moroney

为了使用统计质量保证方法，需要建一张如图 21-2 所示的表格。表中显示 IES、MCC 和 EDR 即是“重要的少数”，它们导致的错误占错误总数的 53%。但是需要注意，在只考虑严重错误时，应该将 IES、EDR、PLT 和 EDL 作为“重要的少数”。一旦确定了这些重要的

456

457

少数原因，软件开发组织就可以开始采取改正行动了。例如，为了改正 MCC 错误，软件开发 者可能要采用需求收集技术（第 8 章），以提高与客户交流和规格说明的质量。为了改正 EDR 错误，开发者可能要使用工具进行数据建模，并进行更为严格的数据设计评审。

总计			严重		中等		微小	
错误	数量	百分比	数量	百分比	数量	百分比	数量	百分比
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
总计	942	100%	128	100%	379	100%	435	100%

图 21-2 统计 SQA 数据收集

值得注意的是，改正行动主要是针对“重要的少数”。随着这些“重要的少数”被不断改正，新的“重要的少数”将提到改正日程上来。

已经证明统计软件质量保证技术确实使质量得到了提高（例如 [Rya11]、[Art97]）。在某些情况下，应用这些技术后，软件组织已经取得每年减少 50% 缺陷的好成绩。

统计 SQA 及 Pareto 原则的应用可以用一句话概括：将时间用于真正重要的地方，但是首先你必须知道什么是真正重要的！

21.6.2 软件工程中的六西格玛

六西格玛是目前产业界应用最广泛的基于统计的质量保证策略。20 世纪 80 年代在摩托罗拉公司最先普及，六西格玛策略“是严格且规范的方法学，它运用数据和统计分析，通过识别和消除制造以及服务相关过程中的‘缺陷’来测量和改进企业的运转状况”[ISI08]。“六西格玛”一词来源于六个标准偏差——每百万个操作发生 3.4 个偏差（缺陷），它意味着非常高的质量标准。六西格玛方法学有三个主要的核心步骤：

- 定义：通过与客户交流的方法来定义客户需求、可交付的产品及项目目标。
- 测量：测量现有的过程及其产品，以确定当前的质量状况（收集缺陷度量信息）。
- 分析：分析缺陷度量信息，并挑选出重要的少数原因。

如果某个现有软件过程是适当的，只是需要改进，则六西格玛还需要另外两个核心步骤：

- 改进：通过消除缺陷根本原因的方式来改进过程。

提问 六西格玛方法学的核心步骤是什么？

● 控制：控制过程以保证以后的工作不会再引入缺陷原因。

以上三个核心步骤和另外两个附加步骤有时叫作 DMAIC (定义、测量、分析、改进和控制) 方法。

如果某组织正在建立一个软件过程 (而不是改进现有的过程), 则需要增加下面两个核心步骤:

- 设计：设计过程, 以达到: (1) 避免缺陷产生的根本原因; (2) 满足客户需求。
- 验证：验证过程模型是否确实能够避免缺陷, 并且满足客户需求。

上述步骤有时称为 DMADV (定义、测量、分析、设计和验证) 方法。

六西格玛的全面讨论不是本书重点, 有兴趣的读者可参考 [ISI08]、[Pyz03] 和 [Sne03]。

458

21.7 软件可靠性

毫无疑问, 计算机程序的可靠性是其整个质量的重要组成部分。如果某个程序经常性且反复性地不能执行, 那么其他软件质量因素是不是可接受就无所谓了。

与其他质量因素不同, 软件可靠性可以通过历史数据和开发数据直接测量和估算出来。按统计术语所定义的软件可靠性是: “在特定环境和特定时间内, 计算机程序正常运行的概率” [Mus87]。举个例子来说, 如果程序 X 在 8 小时处理时间内的可靠性估计为 0.999, 也就意味着, 如果程序 X 执行 1000 次, 每次运行 8 小时处理时间 (执行时间), 则 1000 次中正确运行 (无失效) 的次数可能是 999 次。

引述 可靠性不得以的代价是简明性。

C. A. R. Hoare

无论何时谈到软件可靠性, 都会涉及一个关键问题: 术语失效 (failure) 一词是什么意思? 在有关软件质量和软件可靠性的任何讨论中, 失效都意味着与软件需求的不符。但是这一定义是有等级之分的。失效可能仅仅是令人厌烦的, 也可能是灾难性的。有的失效可以在几秒钟之内得到纠正, 有的则需要几个星期甚至几个月的时间才能纠正。让问题更加复杂的是, 纠正一个失效事实上可能会引入其他的错误, 而这些错误最终又会导致其他的失效。

21.7.1 可靠性和可用性的测量

早期的软件可靠性测量工作试图将硬件可靠性理论中的数学公式外推来进行软件可靠性的预测。大多数与硬件相关的可靠性模型依据的是由于“磨损”而导致的失效, 而不是由于设计缺陷而导致的失效。在硬件中, 由于物理磨损 (如温度、腐蚀、振动的影响) 导致的失效远比与设计缺陷有关的失效多。不幸的是, 软件恰好相反。实际上, 所有软件失效都可以追溯到设计或实现问题上, 磨损 (第 2 章) 在这里根本没有影响。

关键点 软件可靠性问题总能追溯到设计或实现中的缺陷。

硬件可靠性理论中的核心概念以及这些核心概念是否适用于软件, 对这个问题的争论仍然存在。尽管在两种系统之间尚未建立不可辩驳的联系, 但是考虑少数几个同时适用于这两种系统的简单概念却很有必要。

当我们考虑基于计算机的系统时, 可靠性的简单测量是平均失效间隔时间 (Mean-Time-Between-Failure, MTBF):

$$MTBF = MTTF + MTTR$$

其中, MTTF (Mean-Time-To-Failure) 和 MTTR (Mean-Time-To-Repair)

关键点 请注意, 平均失效间隔时间和相关测量是基于 CPU 时间, 而不是挂钟时间。

459

分别是平均失效时间和平均维修时间。^①

许多研究人员认为 MTBF 是一个远比第 30 章讨论的其他与质量有关的软件度量更有用的测量指标。简而言之,最终用户关心的是失效,而不是总缺陷数。由于一个程序中包含的每个缺陷所具有的失效率不同,因此总缺陷数难以表示系统的可靠性。例如,考虑一个程序,已运行 3000 处理器小时没有发生故障。该程序中的许多缺陷在被发现之前可能有数万小时都不会被发现。隐藏如此深的错误的 MTBF 可能是 30000 甚至 60000 处理器小时。其他尚未发现的缺陷,可能有 4000 或 5000 小时的失效率。即使第一类错误(那些具有长时间的 MTBF)都去除了,对软件可靠性的影响也是微不足道的。

然而,MTBF 可能会产生问题的原因有两个:(1)它突出了失效之间的时间跨度,但不会为我们提供一个凸显的失效率;(2)MTBF 可能被误解为平均寿命,即使这不是它的含义。

可靠性的另一个可选衡量指标是失效率(Failures-In-Time, FIT)——一个部件每 10 亿机时发生多少次失效的统计测量。因此,1FIT 相当于每 10 亿机时发生一次失效。

除可靠性测量之外,还应该进行可用性测量。软件可用性是指在某个给定时间点上程序能够按照需求执行的概率。其定义为:

$$\text{可用性} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})} \times 100\%$$

MTBF 可靠性测量对 MTTF 和 MTTR 同样敏感。而可用性测量在某种程度上对 MTTR 较为敏感,MTTR 是对软件可维护性的间接测量。有关软件可靠性测量的更广泛讨论可参看 [Laz11]。

建议 可用性的某些方面(这里不讨论)与失效没有关系。例如,计划停机(用于技术支持功能)也使软件不可用。

21.7.2 软件安全

软件安全是一种软件质量保证活动,它主要用来识别和评估可能对软件产生负面影响并促使整个系统失效的潜在灾难。如果能够在软件过程的早期阶段识别出这些灾难,就可以指定软件设计特性来消除或控制这些潜在的灾难。

引述 民众的安全应是最崇高的法律。

Cicero

建模和分析过程可以视为软件安全的一部分。开始时,根据危险程度和风险高低对灾难进行识别和分类。例如,与汽车上的计算机巡航控制系统相关的灾难可能有:(1)产生失去控制的加速,不能停止;(2)踩下刹车踏板后没有反应(关闭);(3)开关打开后不能启动;(4)减速或加速缓慢。一旦识别出这些系统级的灾难,就可以运用分析技术来确定这些灾难发生的严重性和概率^②。为了达到高效,应该将软件置于整个系统中进行分析。例如,一个微小的用户输入错误(人也是系统的组成部分)有可能会被软件错误放大,产生将机械设备置于不正确位置的控制数据,此时当且仅当外部环境条件满足时,机械设备的不正确位置将引发灾难性的失效。失效树分析、实时逻辑及 Petri 网模型等分析技术 [Eri05] 可以用于预测可能引起灾难的事件链,以及事件链中的各个事件出现的概率。

引述 我无法想象是什么情况可能使该船沉没。现代造船术已经不存在这种可能了。

E. I. Smith

(泰坦尼克船长)

一旦完成了灾难识别和分析,就可以进行软件中与安全相关的需求规格说明了。在规格

① 尽管失效可能需要进行调试(和相关的改正),但在大多数情况下,软件不做任何修改,经过重新启动就可以正常工作。

② 这个方法与第 35 章介绍的风险分析方法类似,主要区别是它注重技术问题,而不是项目相关的问题。

说明中包括一张不希望发生的事件清单,以及针对这些事件所希望产生的系统响应。这样就指明了软件在管理不希望发生的事件方面应起的作用。

尽管软件可靠性和软件安全彼此紧密相关,但是弄清它们之间的微妙差异非常重要。软件可靠性使用统计分析的方法来确定软件失效发生的可能性,而失效的发生未必导致灾难或灾祸。软件安全则考察失效导致灾难发生的条件。也就是说,不能在真空中考虑失效,而是要在整个计算机系统及其所处环境的范围内加以考虑。

软件安全的全面讨论超出了本书的范围,对软件安全和相关系统问题有兴趣的读者可参考 [Fir12]、[Har12]、[Smi05] 和 [Lev95]。

21.8 ISO 9000 质量标准[⊖]

质量保证体系可以定义为:用于实现质量管理的组织结构、责任、规程、过程和资源 [ANS87]。创建质量保证体系的目的是帮助组织以符合规格说明的方式,保证组织的产品和服务满足客户的期望。这些体系覆盖了产品整个生命周期的多种活动,包括计划、控制、测量、测试和报告,并在产品的开发和制造过程中改进质量等级。ISO 9000 标准以通用的术语描述了质量保证体系要素,能够适用于任何行业——不论提供的是何种产品或服务。

某个公司要注册成为 ISO 9000 质量保证体系中的一种模式,该公司的质量体系和实施情况应该由第三方的审核人员仔细检查,查看其是否符合标准以及实施是否有效。成功注册之后,这个公司将获得由审核人员所代表的注册登记实体颁发的证书。此后每半年进行一次监督审核,以此保证该公司的质量体系持续符合标准。

ISO 9001:2008 中描述的质量要求涉及管理者责任、质量体系、合同评审、设计控制、文件和资料控制、产品标识与可追溯性、过程控制、检验和试验、纠正及预防措施、质量记录的控制、内部质量审核、培训、服务以及统计技术等主题。软件组织要登记为 ISO 9001:2008 认证,就必须针对上述每个方面的质量要求(以及其他方面的质量要求)制定相关的政策和规程,并且有能力证明组织活动的确是按照这些政策和规程实施的。希望进一步了解有关 ISO 9001:2008 信息的读者可参考 [Coc11]、[Hoy09] 或 [Cia09]。

网络资源 关于软件安全的有价值的论文集可以在 www.safeware-eng.com 找到。

461

网络资源 大量有关 ISO 9000/9001 资源的链接可在 www.tantara.ab.ca/info.htm 找到。

信息栏 ISO 9001:2008 标准

下面的大纲定义了 ISO 9001:2008 标准的基本要素。与标准有关的详细信息可从国际标准化组织 (www.iso.ch) 及其他网络信息源(如 www.praxiom.com) 找到。

● 建立质量管理体系的要素。

- 建立、实施和改进质量体系。
- 制定质量方针,强调质量体系的重要性。

● 编制质量体系文件。

- 描述过程。
- 编制操作手册。
- 制定控制(更新)文件的方法。
- 制定记录保存的方法。

● 支持质量控制和质量保证。

- 提高所有利益相关者对质量重要性的

⊖ 本节由 Michael Stovsky 编写,根据“Fundamentals of ISO 9000”改编,这是由 R.S.Pressman & Associates,Inc. 为音像教程“Essential Software Engineering”开发的工作手册。使用已得到授权。

认识。

- 注重客户满意度。
- 制定质量计划来描述目的、职责和权力。
- 制定所有利益相关者间的交流机制。
- 为质量管理体系建立评审机制。
 - 确定评审方法和反馈机制。
 - 制定跟踪程序。
- 确定质量资源（包括人员、培训、基础设施要素）。
 - 建立控制机制。
 - 针对计划。
 - 针对客户需求。
 - 针对技术活动（如分析、设计、试验）。
 - 针对项目监测和管理。
 - 制定补救措施。
 - 评估质量数据和度量。
 - 为持续的过程和质量改进制定措施。

462

21.9 软件质量保证计划

SQA 计划为软件质量保证提供了一张路线图。该计划由 SQA 小组（如果没有 SQA 小组，则由软件团队）制定，作为各个软件项目中 SQA 活动的模板。

IEEE 已经公布了 SQA 计划的标准 [IEE93]。该标准建议 SQA 计划应包括：（1）计划的目的和范围；（2）SQA 覆盖的所有软件工作产品的描述（例如，模型、文档、源代码）；（3）应用于软件过程中的所有适用的标准和习惯做法；（4）SQA 活动和任务（包括评审和审核）以及它们在整个软件过程中的位置；（5）支持 SQA 活动和任务的工具和方法；（6）软件配置管理（第 29 章）的规程；（7）收集、保护和维护所有 SQA 相关记录的方法；（8）与产品质量相关的组织角色和责任。

软件工具 软件质量管理

[目标] 使用 SQA 工具的目的是辅助项目团队评估和改进软件工作产品的质量。

[机制] 工具的机制各异。一般情况下，其目的是评估特定工作产品的质量。注意，SQA 工具类通常包含很多软件测试工具（第 22～26 章）。

[代表性工具]^①

- QA Complete。由 SmartBear (<http://smartbear.com/products/qa-tools/test-management>) 开发，QA 管理确保在软件开发过程的每一阶段做到完整的测试覆盖。
- QPR Suite。由 QPR Software (<http://www.qpr.com>) 开发，提供对六西格玛及其他质量管理方法的支持。
- Quality Tools and Templates。由 iSixSigma (<http://www.isixsigma.com/tools-templates/>) 开发，描述了大量有用的质量管理工具和方法。
- NASA Quality Resources。由 Goddard Space Flight Center (<http://www.hq.nasa.gov/office/codeq/software/ComplexElectronics/checklists.htm>) 开发，提供了有用的 SQA 表格、模板、检查单和工具。

① 这里记录的工具体不代表本书支持这些工具，而只是这类工具的例子。在大多数情况下，工具的名字由各白的开发者注册为商标。

21.10 小结

软件质量保证是在软件过程中的每一步都进行的普适性活动。SQA 包括：对方法和工具进行有效应用的规程、对诸如技术评审和软件测试等质量控制活动的监督、变更管理规程、保证符合标准的规程以及测量和报告机制。

为了正确地进行软件质量保证，必须收集、评估和发布有关软件工程过程的数据。基于统计的 SQA 有助于提高产品和软件过程本身的质量。软件可靠性模型将测量加以扩展，能够由所收集的缺陷数据推导出相应的失效率并进行可靠性预测。

总之，应该注意 Dunn 和 Ullman 所说的话 [Dun82]：“软件质量保证就是将质量保证的管理规则和设计规范映射到适用的软件工程管理和技术空间上。”质量保证的能力是成熟的工程学科的尺度。当成功实现上述映射时，其结果就是成熟的软件工程。

习题与思考题

- 21.1 有人说“差异控制是质量控制的核心”，既然每个程序都与其他程序互不相同，我们应该寻找什么样的差异？应该如何控制这些差异？
- 21.2 如果客户不断改变他想做的事情，是否还有可能评估软件质量？
- 21.3 质量和可靠性是两个相关的概念，但在许多方面却有根本的不同，请就此进行讨论。
- 21.4 一个正确的程序还能不可靠吗？请加以解释。
- 21.5 一个正确的程序可能表现不出高质量吗？请加以解释。
- 21.6 为什么软件工程小组与独立的软件质量保证小组之间的关系经常是紧张的？这种紧张关系是否是正常的？
- 21.7 假设赋予你改进组织中软件质量的职责，你要做的第一件事是什么？然后呢？
- 21.8 除了可以统计错误和缺陷之外，还有哪些可以统计的软件特性是具有质量意义的？它们是什么？能否直接测量？
- 21.9 软件中的 MTBF 概念仍然不断受到批评，请解释为什么。
- 21.10 给出两个安全性至关重要的计算机控制系统。并为每个系统列出至少三项与软件失效直接相关的灾难。
- 21.11 获取一份 ISO 9001:2000 和 ISO 9000-3^①的副本，准备一次讨论作业，讨论三个方面的 ISO 9001 质量要求及其在软件行业中是如何应用的。

扩展阅读与信息资源

关于规范的计算机软件质量保证规程，Chemuturi(《Mastering Software Quality Assurance》，J.Ross Publishing,2010)、Hoyle(《Quality Management Essentials》，Butterworth-Heinemann, 2007)、Tian(《Software Quality Engineering》，Wiley-IEEE Computer Society Press, 2005)、El Emam(《The ROI from Software Quality》，Auerbach, 2005)、Horch(《Practical Guide to Software Quality Management》，Artech House, 2003)以及 Nance 和 Arthur(《Managing Software Quality》，Springer, 2002)的书为我们展现了什么是优秀的管理水平。Deming[Dem86]、Defoe 和 Juran(《Juran's Quality Handbook》，6th ed., McGraw-Hill, 2010)、Juran(《Juran on Quality by Design》，Free Press, 1992)以及 Crosby[Cro79]和《Quality Is Still Free》，McGraw-Hill, 1995)的书虽然不是软件方面的书，但是对于负责软件开发的高级经理来说是必读的。Gluckman 和 Roome(《Everyday Heroes of the Quality Movement》，Dorset

① 标准 ISO 9000-3 为 ISO 9001:2000 在计算机软件的使用指南，现已改版为 ISO 90003。——译者注

House, 1993) 通过讲述质量过程中参与者的一个故事, 把质量问题人性化了。Kan (《Metrics and Models in Software Quality Engineering》, Addison-Wesley, 1995) 提出了软件质量的量化观点。

Evans (《Quality & Performance Excellence》, South-Western College Publishing, 2007 和《Total Quality: Management, Organization and strategy》, 4th ed., South-Western College Publishing, 2004)、Bru (《Six Sigma for Managers》, McGraw-Hill, 2005) 和 Dobb (《ISO9001:2000 Quality Registration Step-by-Step》, 3rd ed., Butterworth-Heinemann, 2004) 的书分别是许多关于 TQM、六西格玛和 ISO 9001:2000 的书籍中的代表。

O'Connor 和 Kleyner (《Practical Reliability Engineering》, Wiley, 2012)、Naik 和 Tripathy (《Software Testing and Quality Assurance: Theory and Practice》, Wiley-Spektrum, 2008)、Pham (《System Software Reliability》, Springer, 2006)、Musa (《Software Reliability Engineering: More Reliable Software, Faster Development and Testing》, 2nd ed., McGraw-Hill, 2004) 以及 Peled (《Software Reliability Methods》, Springer, 2001) 撰写了介绍测量和分析软件可靠性方法的实用指南。

Vincoli (《Basic Guide to System Safety》, Wiley, 2006)、Dhillon (《Computer System Reliability, Safety and Usability》, CRC Press, 2013 和《Engineering Safety》, World Scientific Publishing Co., 2003) 以及 Hermann (《Software Safety and Reliability》, Wiley-IEEE Computer Society Press, 2010), 还有 Verma、Ajit 和 Karanki (《Reliability and Safety Engineering》, Springer, 2010)、Storey (《Safety-Critical Computer Systems》, Addison-Wesley, 1996) 以及 Leveson [Lev95] 的书是目前出版的最全的讨论软件安全和系统安全的书籍。此外, van der Meulen (《Definitions for Hardware and Software Safety Engineers》, Springer-Verlag, 2000) 提供了一份完整的可靠性和安全性方面重要概念和术语的汇编, Gardiner (《Testing Safety-Related Software》, Springer-Verlag, 1999) 提供了测试安全关键系统的专业指导。Friedman 和 Voas (《Software Assessment: Reliability Safety and Testability》, Wiley, 1995) 提供了评估可靠性和安全性的有用模型。Ericson (《Hazard Analysis Primer》, CreateSpace Independent Publishing Platform, 2012 和《Hazard Analysis Techniques for System Safety》, Wiley, 2005) 讲述了日益重要领域的公害分析。

网上有软件质量保证和相关主题的大量信息资源。SQA 相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 的“software engineering resources”下找到。

软件测试策略

要点浏览

概念: 软件测试的目的是为了发现软件设计和实现过程中因疏忽所造成的错误。但是, 如何进行测试? 是否应该制定正式的测试计划? 应该将整个程序作为一个整体来测试, 还是应该只测试其中的一小部分? 当向一个大型系统加入新的构件时, 对于已经做过的测试, 是否还要重新测试? 什么时候需要客户参与测试工作? 在制定测试策略时, 就需要回答上述问题以及一些其他问题。

人员: 软件测试策略由项目经理、软件工程师及测试专家来制定。

重要性: 测试所花费的工作量经常比其他任何软件工程活动都多。若测试是无计划进行的, 则既浪费时间, 又浪费不必要的劳动。甚至更糟的是, 错误未被测出, 因而隐蔽下来。因此, 为测试软件建立系统化的测试策略是合情合理的。

步骤: 测试从“小范围”开始, 并逐步过

渡到“软件整体”。这意味着, 早期的测试关注单个构件或相关的一小组构件, 利用测试发现封装在构件中的数据错误和处理逻辑错误。当完成单个构件的测试后, 需要将构件集成起来, 直到建成整个系统。这时, 执行一系列的高阶测试以发现不满足客户需求的错误。随着错误的发现, 必须利用调试过程对错误进行诊断和纠正。

工作产品: 测试规格说明是将软件测试团队的具体测试方法文档化。这主要包括制定描述整体策略的计划, 定义特定测试步骤的规程以及将要进行测试的类型。

质量保证措施: 在测试进行之前通过对测试规格说明进行评审, 可以评估测试用例及测试任务的完整性。有效的测试计划和规程可以引导团队有序地构建软件, 并且在构建过程中能够发现各个阶段引入的错误。

软件测试策略提供了一张路线图: 描述将要进行的测试步骤, 包括这些步骤的计划和执行时机, 以及需要的工作量、时间和资源。因此, 任何测试策略都必须包含测试计划、测试用例设计、测试执行以及测试结果数据的收集与评估。

软件测试策略应该具有足够的灵活性, 以促进测试方法的定制; 同时又必须足够严格, 以支持在项目进行过程中对项目进行合理策划和追踪管理。Shooman[SHO83]对这些问题进行了讨论:

从许多方面来看, 测试都是一个独立的过程, 并且同软件开发方法一样, 测试类型也有很多种。多年以来, 对付程序出错的唯一武器就是谨慎的设计和程序员的个人智慧。目前, 有很多现代设计技术(和正式技术评

关键概念

- α 测试
- β 测试
- 自底向上集成
- 类测试
- 簇测试
- 完成
- 配置评审
- 调试
- 部署测试
- 驱动
- 独立测试组
- 集成测试

审)可以帮助我们减少代码中内在的初始错误。类似地,不同的测试方法正在逐渐聚合成几种不同的途径和思想。

这些途径和思想就是我们所谓的策略。本章讨论软件测试策略,本书的第23~26章介绍实施测试策略的测试方法和测试技术。

22.1 软件测试的策略性方法

测试是可以事先计划并可以系统地进行的一系列活动。因此,应该为软件过程定义软件测试模板,即将特定的测试用例设计技术和测试方法放到一系列的测试步骤中去。

文献中已经提出了许多软件测试策略。这些策略为软件开发人员提供了测试模板,且具备下述一般特征:

- 为完成有效的测试,应该进行有效的、正式的技术评审(第20章)。通过评审,许多错误可以在测试开始之前排除。
- 测试开始于构件层,然后向外“延伸”到整个基于计算机系统的集成中。
- 不同的测试技术适用于不同的软件工程方法和不同的时间点。
- 测试由软件开发人员和(对大型项目而言)独立的测试组执行。
- 测试和调试是不同的活动,但任何测试策略都必须包括调试。

软件测试策略必须提供必要的低级测试,可以验证小段源代码是否正确实现,也要提供高级测试,用来确认系统的主要功能是否满足用户需求。软件测试策略必须为专业人员提供工作指南,同时,为管理者提供一系列的里程碑。由于测试策略的步骤往往是在软件完成的最后期限的压力开始呈现时才刚刚进行的,因此,测试的进度必须是可测量的,并且应该让问题尽可能早地暴露。

22.1.1 验证与确认

软件测试是通常所讲的更为广泛的主题——验证与确认(Verification and Validation, V&V)的一部分。验证是指确保软件正确地实现某一特定功能的一系列活动,而确认指的是确保开发的软件可追溯到客户需求的另外一系列活动。Boehm[BOE81]用另一种方式说明了这两者的区别:

验证:“我们在正确地构建产品吗?”

确认:“我们在构建正确的产品吗?”

验证与确认的定义包含很多软件质量保证活动(第21章)^①。

验证与确认包含广泛的SQA活动:正式技术评审、质量和配置审核、性能监控、仿真、可行性研究、文档评审、数据库评审、算法分析、开发测试、易用性测试、合格性测试、验收测试和安装测试。虽然测试在验证

关键概念

面向对象软件
性能测试
恢复测试
回归测试
安全测试
冒烟测试
压力测试
桩
系统测试
移动App的测试策略
WebApp的测试策略
基于线程的测试
自顶向下集成
单元测试确认
确认测试
验证

网络资源

有关软件测试的有用资料可在www.mtsu.edu/~storm/找到。

引述

测试是开发软件系统过程中每项可靠的工作都不可避免的部分。

William Howden

^① 应该注意到,哪些类型的测试构成“确认”,然而人们对此观点存在极大的分歧。一些人认为所有的测试都是验证,而确认是在对需求进行评审和认可时进行的,也许更晚一些——是在系统投入运行时由用户进行的。另外一些人将单元测试和集成测试(22.3.1节和22.3.2节)看成验证,而将高阶测试(22.6节和22.7节)看成确认。

与确认中起到了非常重要的作用，但很多其他活动也是必不可少的。

测试确实为软件质量的评估（更实际地说是错误的发现）提供了最后的堡垒。但是，测试不应当被看作安全网。正如人们所说的那样：“你不能测试质量。如果开始测试之前质量不佳，那么当你完成测试时质量仍然不佳。”在软件工程的整个过程中，质量已经被包含在软件之中了。方法和工具的正确运用、有效的正式技术评审、坚持不懈的管理与测量，这些都形成了在测试过程中所确认的质量。

Miller[MIL77]将软件测试和质量保证联系在一起，他认为：“无论是大规模系统还是小规模系统，程序测试的根本动机都是使用经济且有效的方法来确认软件质量。”

建议 不要轻易地将测试看成是一个安全网，认为它能捕捉由不良的软件工程实践引发的所有错误。应该在整个软件过程中注重质量和错误检测。

22.1.2 软件测试组织

对每个软件项目而言，在测试开始时就会存在固有的利害关系冲突。要求开发软件的人员对该软件进行测试，这本身似乎是没有恶意的！毕竟，谁能比开发者本人更了解程序呢？遗憾的是，这些开发人员感兴趣的是急于显示他们所开发的程序是无错误的，是按照客户的需求开发的，而且能按照预定的进度和预算完成。这些利害关系会影响软件的充分测试。

引述 乐观主义是编程的职业障碍；测试是治疗良方。

Kent Beck

468

从心理学的观点来看，软件分析和设计（连同编码）是建设性的任务。软件工程师分析、建模，然后编写计算机程序及其文档。与其他任何建设者一样，软件工程师也为自己的“大厦”感到骄傲，而蔑视企图拆掉大厦的任何人。当测试开始时，有一种微妙的但确实存在的企图，即试图摧毁软件工程师所建造的大厦。以开发者的观点来看，可以认为（心理学上）测试是破坏性的。因此，开发者精心地设计和执行测试，试图证明其程序的正确性，而不是注意发现错误。遗憾的是，错误仍然是存在的，而且，即使软件工程师没有找到错误，客户也会发现它们。

上述讨论通常会使人产生下面的误解：（1）软件开发人员根本不应该做测试；（2）应当让那些无情地爱挑毛病的陌生人做软件测试；（3）测试人员仅在测试步骤即将开始时参与项目。这些想法都是不正确的。

软件开发人员总是要负责程序各个单元（构件）的测试，确保每个单元完成其功能或展示所设计的行为。在多数情况下，开发者也进行集成测试。集成测试是一个测试步骤，它将给出整个软件体系结构的构建（和测试）。只有在软件体系结构完成后，独立测试组才开始介入。

独立测试组（Independent Test Group, ITG）的作用是为了避免开发人员进行测试所引发的固有问题。独立测试可以消除利益冲突。独立测试组的成员毕竟是依靠找错误来获得报酬的。

关键点 独立测试组没有软件开发人员可能经历的“利益冲突”。

然而，软件开发人员并不是将程序交给独立测试组就可以一走了之。在整个软件项目中，开发人员和测试组要密切配合，以确保进行充分的测试。在测试进行的过程中，必须随时可以找到开发人员，以便及时修改发现的错误。

引述 人们犯的第一个错误是认为测试团队负责保证质量。

Brian Marick

从分析与设计到计划和制定测试规程，ITG参与整个项目过程。从这种意义上讲，ITG是软件开发项目团队的一部分。然而，在很多情况下，

ITG 直接向软件质量保证组织报告，由此获得一定程度的独立性。如果 ITG 是软件工程团队的一部分，那么这种独立性将是不可能获得的。

22.1.3 软件测试策略——宏观

可以将软件过程看作图 22-1 所示的螺旋。开始时系统工程定义软件的角色，从而引出软件需求分析，在需求分析中建立了软件的信息域、功能、行为、性能、约束和确认标准。沿着螺旋向内，经过设计阶段，最后到达编码阶段。为开发计算机软件，沿着流线螺旋前进，每走一圈都会降低软件的抽象层次。

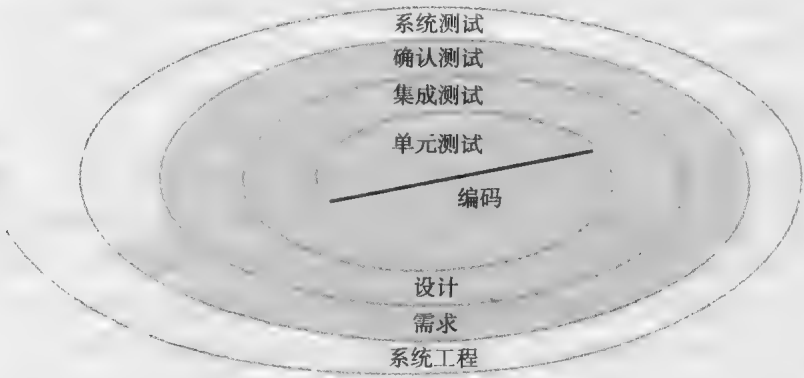


图 22-1 测试策略

软件测试策略也可以放在螺旋模型中来考虑（图 22-1）。单元测试起始于螺旋的旋涡中心，侧重于以源代码形式实现的每个单元（例如，构件、类或 WebApp 内容对象）。沿着螺旋向外就是集成测试，这时的测试重点在于软件体系结构的设计和构建。沿着螺旋向外再走一圈就是确认测试，在这个阶段，依据已经建立的软件，对需求（作为软件需求建模的一部分而建立）进行确认。最后到达系统测试阶段，将软件与系统的其他成分作为一个整体来测试。为了测试计算机软件，沿着流线向外螺旋前进，每转一圈都拓宽了测试范围。

提问 什么是软件测试的总体策略？

以过程的观点考虑整个测试过程，软件工程环境中的测试实际上就是按顺序实现四个步骤，如图 22-2 所示。最初，测试侧重于单个构件，确保它起到了单元的作用，因此称之为单元测试。单元测试充分利用测试技术，运行构件中每个控制结构的特定路径，以确保路径的完全覆盖，并最大限度地发现错误。接下来，组装或集成各个构件以形成完整的软件包。集成测试处理并验证与程序构建相关的问题。在集成过程中，普遍使用关注输入和输出的测试用例设计技术（尽管也使用检验特定程序路径的测试用例设计技术来保证主要控制路径的覆盖）。在软件集成（构建）完成之后，要执行一系列的高阶测试。必须评估确认准则（需求分析阶段建立的）。确认测试为软件满足所有的功能、行为和性能需求提供最终保证。

网络资源 有关软件测试人员的有用资源可在站点 www.SQAtester.com 中找到。

最后的高阶测试步骤已经超出软件工程的边界，属于更为广泛的计算机系统工程范围。软件一旦确认，就必须与其他系统成分（如硬件、人、数据库）结合在一起。系统测试验证所有成分都能很好地结合在一起，且能满足整个系统的功能或性能需求。

470

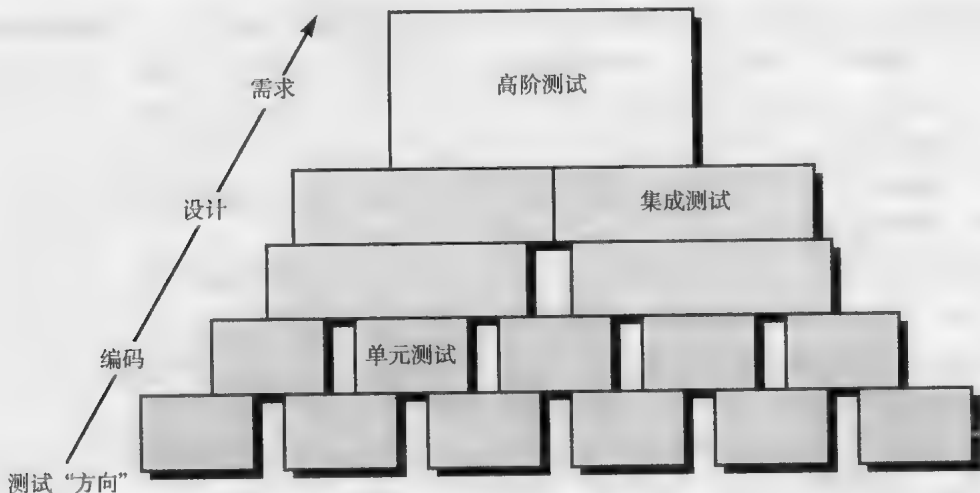


图 22-2 软件测试步骤

SafeHome

准备测试

[场景] Doug Miller 的办公室，继续构件级设计，并开始特定构件的构建。

[人物] Doug Miller，软件工程经理；Vinod、Jamie、Ed 和 Shakira，SafeHome 软件工程团队成员。

[对话]

Doug: 在我看来，我们似乎是没有花费太多的时间讨论测试。

Vinod: 对，但我们都有点忙。另外，我们一直在考虑这个问题……实际上，远不止考虑。

Doug (微笑): 我知道，大家都在超负荷地工作，不过我们还得全面考虑。

Shakira: 在对构件开始编码之前，我喜欢设计单元测试，因此，那是我一直尽力去做的。我有一个相当大的测试文件，一旦完成了构件编码工作，就运行这个测试文件。

Doug: 那是极限编程（敏捷软件开发过程，

见第 5 章）概念，不是吗？

Ed: 是的。尽管我没有亲自使用极限编程，但可以肯定，在建立构件之前设计单元测试是个好主意，这种单元测试的设计会给我们提供所需要的所有信息。

Jamie: 我一直在做这件事情。

Vinod: 我负责集成，因此，每当别人将构件传给我，我就将其集成到部分已集成的程序中，并运行一系列的集成测试。我一直忙于为系统中的每个功能设计适当的测试集。

Doug (对 Vinod): 你多长时间运行一次测试？

Vinod: 每天……直到系统被集成……嗯，直到我们计划交付的软件增量被集成。

Doug: 你们已经走在我前面了。

Vinod (大笑): 在软件业务中，抢先就是一切，老板。

22.1.4 测试完成的标准

每当讨论软件测试时，就会引出一个典型的问题：“测试什么时候才算做完？怎么知道我们已做了足够的测试？”非常遗憾的是，这个问题没

提问 我们什么时候完成测试？

有确定的答案，只是有一些实用的答复和早期的尝试可作为经验指导。

对上述问题的一个答复是：你永远也不能完成测试，这个担子只会从你（软件工程师）身上转移到最终用户身上。用户每次运行计算机程序时，程序就在经受测试。这个严酷的事实突出了其他软件质量保证活动的重要性。另一个答复（有点讽刺意味，但无疑是准确的）是：当你的时间或资金耗尽时，测试就完成了。

尽管很少有专业人员对上面的答复有异议，但软件工程师还是需要更严格的标准，以确定充分的测试何时能做完。净室软件工程方法（第28章）提出了统计使用技术 [ke100]：运行从统计样本中导出的一系列测试，统计样本来自目标群的所有用户对程序的所有可能执行。通过在软件测试过程中收集度量数据并利用现有的统计模型，对于回答“测试何时做完”这种问题，还是有可能提出有意义的指导性原则的。

引述 仅仅针对最终用户需求进行测试，就如同在牺牲了地基、大梁及管道工程的情况下，只根据内部设计师已经完成的工作对建筑进行检查一样。

Boris Beizer

22.2 策略问题

本章的后面几节介绍系统化的软件测试策略。然而，如果忽视了一些重要问题，即使最好的策略也会失败。Tom Gilb[GIL95] 提出，只有软件测试人员解决了下述问题，软件测试策略才会获得成功：（1）早在开始测试之前，就要以量化的方式规定产品需求；（2）明确地陈述测试目标；（3）了解软件的用户并为每类用户建立用户描述；（4）制定强调“快速周期测试”的测试计划；^①（5）建立能够测试自身的“健壮”软件（防错技术在 22.3.1 节讨论）；（6）测试之前，利用有效的正式技术评审作为过滤器；（7）实施正式技术评审以评估测试策略和测试用例本身；（8）为测试过程建立一种持续的改进方法（第 37 章）。

提问 什么样的指导原则使软件测试策略获得成功？

网络资源 相当好的测试资源列表可在 www.SQAtester.com 找到。

22.3 传统软件的测试策略^②

许多策略可用于测试软件。其中的一个极端是，软件团队等到系统完全建造后再对整个系统执行测试，以期望发现错误。虽然这种方法很有吸引力，但效果不好，可能得到的是有许多缺陷的软件，致使所有的利益相关者感到失望。另一个极端是，无论系统的任何一部分在何时建成，软件工程师每天都在进行测试。

多数软件团队选择介于这两者之间的测试策略。这种策略以渐进的观点对待测试，以个别程序单元的测试为起点，逐步转移到便于单元集成的测试（有的时候每天都进行测试），最后以实施整个系统的测试而告终。下面几节将对这几种不同的测试进行描述。

建议 在为构件开发代码之前就设计单元测试用例是个不错的想法，有助于确保开发的代码能够通过测试。

22.3.1 单元测试

单元测试侧重于软件设计的最小单元（软件构件或模块）的验证工作。利用构件级设计

① Gilb[Gil95] 建议软件团队学习对客户可用性进行快速周期测试（项目工作的 2%），至少在“可试验性”方面增强功能性或提高质量。从这些快速周期测试得到的反馈可以用于控制质量级别及相应的测试策略。

② 本书使用术语常规软件和传统软件来表示在多种应用领域中经常碰到的普通分层软件体系结构或调用-返回软件体系结构。传统的软件体系结构不是面向对象的，也不包括 WebApp 或移动 App。

描述作为指南，测试重要的控制路径以发现模块内的错误。测试的相对复杂度和这类测试发现的错误受到单元测试约束范围的限制。单元测试侧重于构件的内部处理逻辑和数据结构。这种类型的测试可以对多个构件并行执行。

单元测试问题。图 22-3 对单元测试进行了概要描述。测试模块的接口是为了保证被测程序单元的信息能够正常地流入和流出；检查局部数据结构以确保临时存储的数据在算法的整个执行过程中能维持其完整性；执行控制结构中的所有独立路径（基本路径）以确保模块中的所有语句至少执行一次；测试边界条件确保模块在到达边界值的极限或受限处理的情形下仍能正确执行。最后，要对所有的错误处理路径进行测试。

对穿越模块接口的数据流的测试要在任何其他测试开始之前进行。若数据不能正确地输入/输出，则其他测试都是没有意义的。另外，应当测试局部数据结构，可能的话，在单元测试期间确定对全局数据的局部影响。

在单元测试期间，选择测试的执行路径是最基本的任务。设计测试用例是为了发现因错误计算、不正确的比较或不适当的控制流而引起的错误。

边界测试是最重要的单元测试任务之一。软件通常在边界处出错，也就是说，错误行为往往出现在处理 n 维数组的第 n 个元素，或者 i 次循环的第 i 次调用，或者遇到允许出现的最大、最小数值时。使用刚好小于、等于或大于最大值和最小值的数据结构、控制流和数值作为测试用例就很有可能发现错误。

好的设计要求能够预置出错条件并设置异常处理路径，以便当错误确实出现时重新确定路径或彻底中断处理。Yourdon[YOU75] 称这种方法为防错法（antibugging）。遗憾的是，存在的一种趋势是在软件中引入异常处理，然而却从未对其进行测试。如果已经实现了错误处理路径，就一定要对其进行测试。

在评估异常处理时，应能测试下述的潜在错误：（1）错误描述难以理解；（2）记录的错误与真正遇到的错误不一致；（3）在异常处理之前，错误条件就引起了操作系统的干预；（4）异常条件处理不正确；（5）错误描述没有提供足够的信息，对确定错误产生原因没有帮助。

单元测试过程。单元测试通常被认为是编码阶段的附属工作。可以在编码开始之前或源代码生成之后进行单元测试的设计。设计信息的评审可以指导建立测试用例，发现前面所讨论的各类错误，每个测试用例都应与一组预期结果联系在一起。

由于构件并不是独立的程序，因此，必须为每个测试单元开发驱动程序和桩程序。单元测试环境如图 22-4 所示。在大多数应用中，驱动程序只是一个“主程序”，它接收测试用例数据，将这些数据传递给（将要测试的）构件，并打印相关结果。桩程序的作用是替换那些从属于被测构件（或被其调用）的模块。桩程序或“伪程序”使用从属模块的接口，可能做少量



图 22-3 单元测试

473

提问 单元测试期间常发现的错误是什么？

建议 确信已经设计了执行每个异常处理路径的测试。若没有，当执行这样的路径时就可能失败，从而加重了危险的形势。

474

建议 在没有资源做全面测试的情况下，只选择关键模块和高复杂性模块做单元测试。

的数据操作，提供入口的验证，并将控制返回到被测模块。

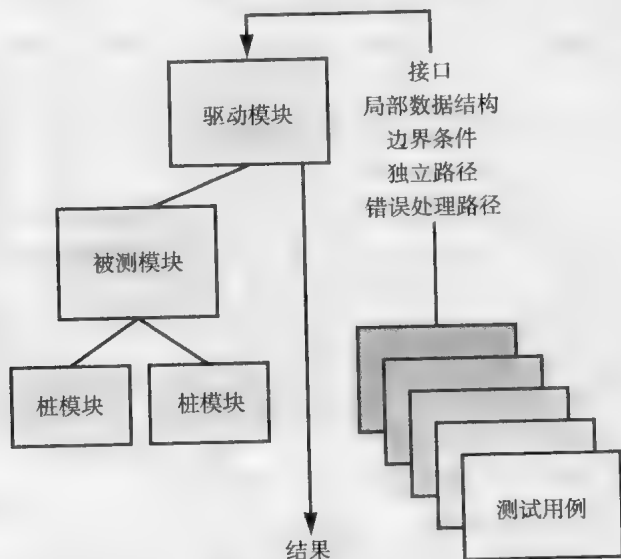


图 22-4 单元测试环境

驱动程序和桩程序都意味着测试开销。也就是说，两者都必须编写代码（通常并没有使用正式的设计），但并不与最终的软件产品一起交付。若驱动程序和桩程序保持简单，实际开销就会比较低。遗憾的是，在只使用“简单”的驱动程序和桩程序的情况下，许多构件是不能完成充分的单元测试的，因此，完整的测试可以延迟到集成测试这一步（这里也要使用驱动程序和桩程序）。

22.3.2 集成测试

软件界的初学者一旦完成所有模块的单元测试之后，可能会问一个似乎很合理的问题：如果每个模块都能单独工作得很好，那么为什么要怀疑将它们放在一起时的工作情况呢？当然，这个问题涉及“将它们放在一起”的接口连接。数据可能在穿过接口时丢失；一个模块可能对另一个模块产生负面影响；子功能联合在一起并不能达到预期的功能；单个模块中可以接受的不精确性在连接起来之后可能会扩大到无法接受的程度；全局数据结构可能产生问题。遗憾的是，问题还远不止这些。

建议 采取一步到位的集成方法是一种懒惰的策略，注定会失败。在进行测试时，应该采用增量集成。

集成测试是构建软件体系结构的系统化技术，同时也是进行一些旨在发现与接口相关的错误的测试。其目标是利用已通过单元测试的构件建立设计中描述的程序结构。

常常存在一种非增量集成的倾向，即利用“一步到位”的方式来构造程序。所有的构件都事先连接在一起，全部程序作为一个整体进行测试。结果往往是一片混乱！会出现一大堆错误。由于在整个程序的广阔区域中分离出错的原因是非常复杂的，因此改正错误也会比较困难。

增量集成与“一步到位”的集成方法相反。程序以小增量的方式逐步进行构建和测试，这样错误易于分离和纠正，更易于对接口进行彻底测试，而且可以运用系统化的测试方法。下面将讨论一些不同的增量集成策略。

自顶向下集成。自顶向下集成测试是一种构建软件体系结构的增量方法。模块的集成顺

序为从主控模块（主程序）开始，沿着控制层次逐步向下，以深度优先或广度优先的方式将从属于（和间接从属于）主控模块的模块集成到结构中去。

参见图 22-5，深度优先集成是首先集成位于程序结构中主控路径上的所有构件。主控路径的选择有一点武断，也可以根据特定应用的特征进行选择。例如，选择最左边的路径，首先集成构件 M_1 、 M_2 和 M_5 。其次，集成 M_8 或 M_6 （若 M_2 的正常运行是必需的），然后集成中间和右边控制路径上的构件。广度优先集成首先沿着水平方向，将属于同一层的构件集成起来。如图 22-5 中，首先将构件 M_2 、 M_3 和 M_4 集成起来，其次是下一个控制层 M_5 、 M_6 ，依此类推。集成过程可以通过下列 5 个步骤完成：

- 1. 主控模块用作测试驱动模块，用直接从属于主控模块的所有模块代替桩模块。
- 2. 依靠所选择的集成方法（深度优先或广度优先），每次用实际模块替换一个从属桩模块。
- 3. 集成每个模块后都进行测试。
- 4. 在完成每个测试集之后，用实际模块替换另一个桩模块。
- 5. 可以执行回归测试（在本节的后面讨论）以确保没有引入新的错误。回到第 2 步继续执行此过程，直到完成了整个程序结构的构建。

建议 制定项目进度时，必须考虑将要采取的集成方式，使得构件在需要时是可用的。

提问 自顶向下集成的步骤是什么？

476

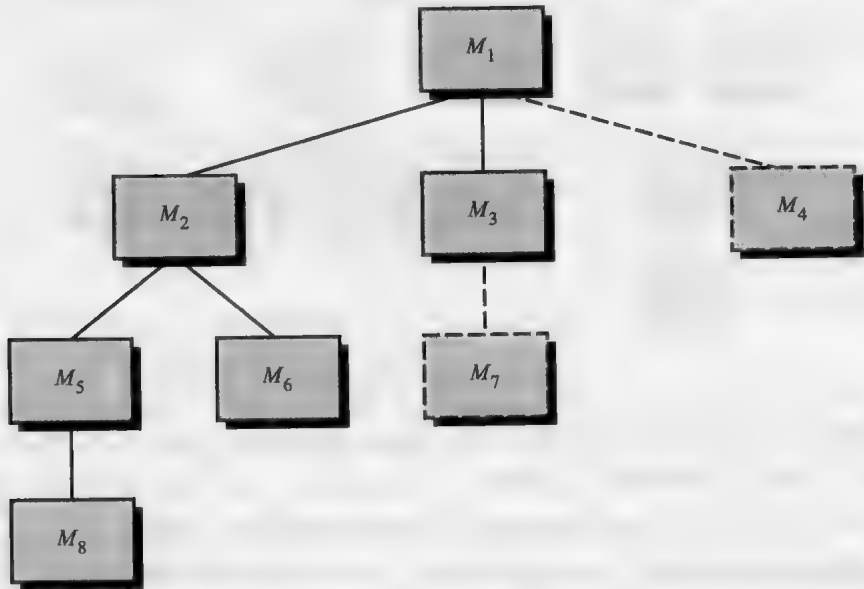


图 22-5 自顶向下集成

自顶向下集成策略是在测试过程的早期验证主要控制点或决策点。在能够很好分解的程序结构中，决策发生在层次结构的较高层，因此会首先遇到。如果主控问题确实存在，尽早地发现是有必要的。若选择了深度优先集成方法，可以实现和展示软件的某个完整功能。较早的功能展示可以增强所有开发者、投资者及用户的信心。

自底向上集成测试。自底向上集成测试，顾名思义，就是从原子模块（程序结构的最底层构件）开始进行构建和测试。由于构件是自底向上集成的，在处理时所需要的从属于给定层次的模块总是存在的，因此，没有必

提问 选择自顶向下集成方法时，可能会遇到什么问题？

要使用桩模块。自底向上集成策略可以利用以下步骤来实现：

- 1. 连接低层构件以构成完成特定子功能的簇（有时称为构造）。
- 2. 编写驱动模块（测试的控制程序）以协调测试用例的输入和输出。
- 3. 测试簇。
- 4. 去掉驱动程序，沿着程序结构向上逐步连接簇。

477

遵循这种模式的集成如图 22-6 所示。连接相应的构件形成簇 1、簇 2 和簇 3，利用驱动模块（图中的虚线框）对每个簇进行测试。簇 1 和簇 2 中的构件从属于模块 M_a ，去掉驱动模块 D_1 和 D_2 ，将这两个簇直接与 M_a 相连。与之相类似，在簇 3 与 M_b 连接之前去掉驱动模块 D_3 。最后将 M_a 和 M_b 与构件 M_c 连接在一起，依此类推。

提问 自底向上的集成步骤是什么？

关键点 自底向上集成排除了对复杂桩的需要。

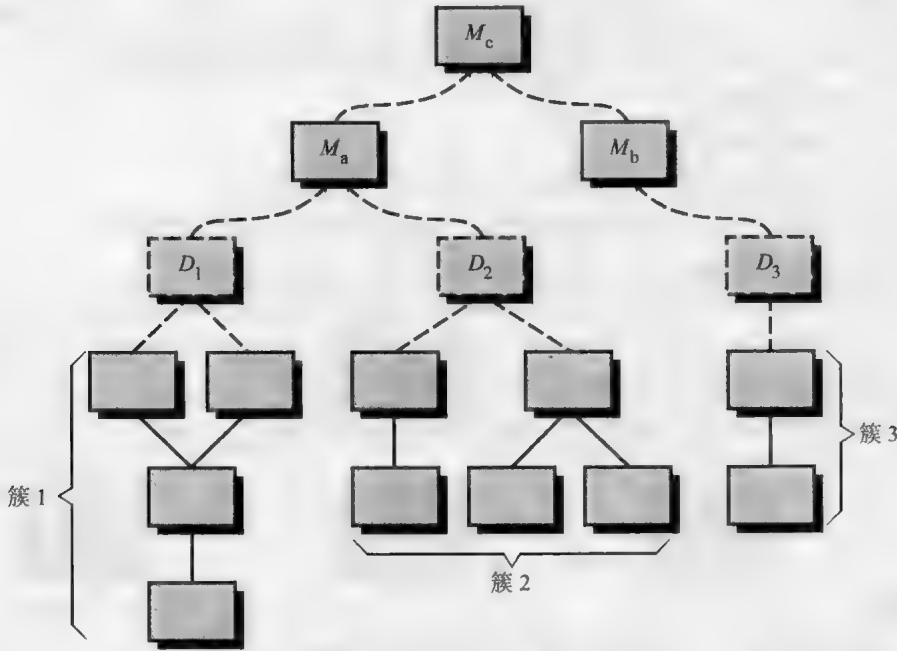


图 22-6 自底向上集成

随着集成的向上进行，对单独的测试驱动模块的需求减少。事实上，若程序结构的最上两层是自顶向下集成的，则驱动模块的数量可以大大减少，而且簇的集成得到了明显简化。

回归测试。每当加入一个新模块作为集成测试的一部分时，软件发生变更，建立了新的数据流路径，可能出现新的 I/O，还可能调用新的控制逻辑。这些变更所带来的副作用可能会使原来可以正常工作的功能产生问题。在集成测试策略的环境下，回归测试是重新执行已测试过的某些子集，以确保变更没有传播不期望的副作用。回归测试有助于保证变更（由于测试或其他原因）不引入无意识行为或额外的错误。

建议 回归测试是减少“副效应”的重要方法。每次对软件做重要变更时（包括新构件的集成），都要进行回归测试。

回归测试可以手工进行，方法是重新执行所有测试用例的子集，或者利用捕捉/回放工具自动进行。捕捉/回放工具使软件工程师能够为后续的回放与比较捕捉测试用例和测试结果。回归测试套件（将要执行的测试子集）包含以下三种测试用例：

478

- 能够测试软件所有功能的具有代表性的测试样本。
- 额外测试，侧重于可能会受变更影响的软件功能。

- 侧重于已发生变更的软件构件测试。

随着集成测试的进行,回归测试的数量可能变得相当庞大,因此,应将回归测试套件设计成只包括涉及每个主要程序功能的一个或多个错误类的测试。

冒烟测试。开发软件产品时,冒烟测试是一种常用的集成测试方法,是时间关键项目的决定性机制,允许软件团队频繁地对项目进行评估。大体上,冒烟测试方法包括下列活动。

1. 将已经转换为代码的软件构件集成到构造 (build) 中。一个构造包括所有的数据文件、库、可复用的模块以及实现一个或多个产品功能所需的工程化构件。
2. 设计一系列测试以暴露影响构造正确地其功能的错误。其目的是发现极有可能造成项目延迟的业务阻塞 (show stopper) 错误。
3. 每天将该构造与其他构造及整个软件产品 (以其当前的形式) 集成起来进行冒烟测试。这种集成方法可以是自顶向下的,也可以自底向上的。

每天频繁的测试让管理者和专业人员都能够对集成测试的进展做出实际的评估。McConnell[MCO96] 是这样描述冒烟测试的:

冒烟测试应该对整个系统进行彻底的测试。它不一定是穷举的,但应能暴露主要问题。冒烟测试应该足够彻底,以使得若构造通过测试,则可以假定它足够稳定以致能经受更彻底的测试。

当应用于复杂的、时间关键的软件工程项目时,冒烟测试提供了下列好处:

- 降低了集成风险。冒烟测试是每天进行的,能较早期地发现不相容性和业务阻塞错误,从而降低了因发现错误而对项目进度造成严重影响的可能性。
- 提高最终产品的质量。由于这种方法是面向构建 (集成) 的,因此,冒烟方法既有可能发现功能性错误,也有可能发现体系结构和构件级设计错误。若较早地改正了这些错误,产品的质量就会更好。
- 简化错误的诊断和修正。与所有的集成测试方法一样,冒烟测试期间所发现的错误可能与新的软件增量有关,也就是说,新发现的错误可能来自刚加入到构造中的软件。
- 易于评估进展状况。随着时间的推移,更多的软件被集成,也更多地展示出软件的工作状况。这就提高了团队的士气,并使管理者对项目进展有较好的把握。

集成测试工作产品。软件集成的总体计划和特定的测试描述应该在测试规格说明中文档化。这项工作产品包含测试计划和测试规程,并成为软件配置的一部分。测试可以分为若干个阶段和处理软件特定功能及行为特征的若干个构造来实施。例如, SafeHome 安全系统的集成测试可以划分为以下测试阶段:用户交互,传感器处理,通信功能及警报处理。

每个集成测试阶段都刻画了软件内部广泛的功能类别,而且通常与软件体系结构中特定的领域相关,因此,对应于每个阶段建立了相应的程序构造 (模块集)。

集成的进度、附加的开发以及相关的问题也在测试计划中讨论。确定每个阶段的开始和结束时间,定义单元测试模块的“可用性窗口”。附加软件 (桩模块及驱动模块) 的简要描述侧重于可能需要特殊工作的特征。最后,描述测试环境和资源。特殊的硬件配置、特殊的仿

关键点 冒烟测试被称为是一种滚动的集成测试方法。每天对软件进行重构 (加入新的构件), 并进行冒烟测试。

引述 将每日构造当成项目的心跳。如果没有了心跳,项目就死了。

Jim McCarthy

提问 从冒烟测试中可以得到什么好处?

真器和专门的测试工具或技术也是需要讨论的问题。

紧接着需要描述的是实现测试计划所必需的详细测试规程。描述集成的顺序以及每个集成步骤中对应的测试，其中也包括所有的测试用例（带注释以便后续工作参考）和期望的结果列表。

实际测试结果、问题或特例的历史要记录在测试报告中，若需要的话可附在测试规格说明后面。这部分包含的信息在软件维护期间很重要。也要给出适当的参考文献和附录。

22.4 面向对象软件的测试策略^①

简单地说，测试的目标就是在现实的时间范围内利用可控的工作量找到尽可能多的错误。对于面向对象软件，尽管这个基本目标是不变的，但面向对象软件的本质特征改变了测试策略和测试战术（第 24 章）。

22.4.1 面向对象环境中的单元测试

在考虑面向对象的软件时，单元的概念发生了变化。封装导出了类和对象的定义。这意味着每个类和类的实例包装有属性（数据）和处理这些数据的操作。封装的类常是单元测试的重点，然而，类中包含的操作（方法）是最小的可测试单元。由于类中可以包含很多不同的操作，且特殊的操作可以作为不同类的一部分存在，因此，必须改变单元测试的战术。

我们不再孤立地对单个操作进行测试（传统的单元测试观点），而是将其作为类的一部分。为便于说明，考虑一个类层次结构，在此结构内对超类定义某操作 X ，并且一些子类继承了操作 X 。每个子类使用操作 X ，但它应用于为每个子类定义的私有属性和操作的环境内。由于操作 X 应用的环境有细微的差别，因此有必要在每个子类的环境中测试操作 X 。这意味着在面向对象环境中，以独立的方式测试操作 X （传统的单元测试方法）往往是无效的。

关键点 面向对象的类测试与传统软件的模块测试相似。对操作进行孤立测试是不可取的。

面向对象软件的类测试等同于传统软件的单元测试。不同的是传统软件的单元测试侧重于模块的算法细节和穿过模块接口的数据，而面向对象软件的类测试是由封装在该类中的操作和类的状态行为驱动的。

22.4.2 面向对象环境中的集成测试

由于面向对象软件没有明显的层次控制结构，因此，传统的自顶向下和自底向上集成策略（22.3.2 节）已没有太大意义。另外，由于类的成分间直接或间接的相互作用，因此每次将一个操作集成到类中（传统的增量集成方法）往往是不可能的 [Ber93]。

关键点 面向对象软件集成测试的一个重要策略是基于线程的测试。线程是对一个输入或事件做出反应的类集合。基于使用的测试侧重于那些不与其他类进行频繁协作的类。

面向对象系统的集成测试有两种不同的策略 [Bin94b]。一种策略是基于线程的测试（thread-based testing），对响应系统的一个输入或事件所需的一组类进行集成。每个线程单独地集成和测试。应用回归测试以确保没有产生副作用。另一种方法是基于使用的测试（use-based testing），通过测试很少使用服务类（如果有的话）的那些类（称为独立类）开始系统的构

① 基本的面向对象概念在附录 2 中介绍。

建。独立类测试完成后，利用独立类测试下一层次的类（称为依赖类）。继续依赖类的测试直到完成整个系统。

在进行面向对象系统的集成测试时，驱动模块和桩模块的使用也发生了变化。驱动模块可用于低层操作的测试和整组类的测试。驱动模块也可用于代替用户界面，以便在界面实现之前就可以进行系统功能的测试。桩模块可用于类间需要协作但其中的一个或多个协作类还未完全实现的情况。

簇测试（cluster testing）是面向对象软件集成测试中的一个步骤。这里，借助试图发现协作错误的测试用例来测试（通过检查 CRC 和对象关系模型所确定的）协作的类簇。

22.5 WebApp 的测试策略

WebApp 测试策略采用所有软件测试的基本原理，并使用面向对象系统所使用的策略和战术。此方法可概括为：

1. 对 WebApp 的内容模型进行评审，以发现错误。
2. 对接口模型进行评审，保证适合所有的用例。
3. 评审 WebApp 的设计模型，发现导航错误。
4. 测试用户界面，发现表现机制和导航机制中的错误。
5. 对每个功能构件进行单元测试。
6. 对贯穿体系结构的导航进行测试。
7. 在各种不同的环境配置下实现 WebApp，并测试 WebApp 对于每一种配置的兼容性。
8. 进行安全性测试，试图攻击 WebApp 或其所处环境的弱点。
9. 进行性能测试。
10. 通过可监控的最终用户群对 WebApp 进行测试，对他们与系统的交互结果进行错误评估。

因为很多 WebApp 在不断进化，所以 WebApp 测试是 Web 支持人员所从事的一项持续活动，他们使用回归测试，这些测试是从首次开发 WebApp 时所开发的测试中导出的。WebApp 测试方法将在第 25 章讨论。

关键点 WebApp 测试的总体策略在这里可以总结为 10 个步骤。

网络资源 Web-App 测试方面的优秀文章可以在 www.sticky-minds.com/test-ing.asp 找到。

482

22.6 移动 App 的测试策略

移动 App 的测试策略采用所有软件测试的基本原则。然而，移动 App 的独特性质要求考虑一些特殊的测试方法：

- 用户体验测试。用户在开发过程的早期就介入，以确保移动 App 在所有被支持的设备上实现其可用性及利益相关者的最高期望。
- 设备兼容性测试。测试人员要验证移动 App 在所有要求的硬件设备和软件组合上都能够正确地工作。
- 性能测试。测试人员要专门针对移动设备检查非功能性需求（例如，下载时间、处理器速度、存储容量、供电）。
- 连接性测试。测试人员要确保移动 App 能够访问任何所需要的网络或者 Web 服务，并且可以容忍微弱或者中断的网络访问。

- 安全性测试。测试人员要确保移动 App 没有违背用户的私密性及安全性需求。
 - 在现实环境中测试。现实条件下, 在实际用户的设备上及全球各种网络环境中对 App 进行测试。
 - 认证测试。测试人员要确保移动 App 满足发布它的应用程序商店所建立的标准。
- 移动 App 的测试方法将在第 26 章讨论。

22.7 确认测试

确认测试始于集成测试的结束, 那时已测试完单个构件, 软件已组装成完整的软件包, 且接口错误已被发现和改正。在进行确认测试或系统级测试时, 不同类型软件之间的差别已经消失, 测试便集中于用户可见的动作和用户可识别的系统输出。

确认可用几种方式进行定义, 但是, 其中一个简单 (尽管粗糙) 的定义是当软件可以按照客户合理的预期方式工作时, 确认就算成功。在这一点上, 喜欢吹毛求疵的软件开发人员可能会提出异议: “谁或者什么是合理预期的裁决者呢?” 如果已经开发了软件需求规格说明文档, 那么此文档就描述了所有用户可见的软件属性, 并包含确认准则部分, 确认准则部分就形成了确认测试方法的基础。

关键点 与所有其他测试步骤类似, 确认测试尽力发现错误, 但是它侧重于需求级的错误, 即那些对最终用户而言显而易见的错误。

22.7.1 确认测试准则

软件确认是通过一系列表明软件功能与软件需求相符合的测试而获得的。测试计划列出将要执行的测试类, 测试规程定义了特定的测试用例, 设计的特定测试用例用于确保软件满足所有的功能需求, 具有所有的行为特征, 所有内容都准确无误且正确显示, 达到所有的性能需求, 文档是正确的、可用的, 且满足其他需求 (如可移植性、兼容性、错误恢复和可维护性)。如果发现了与规格说明的偏差, 则要创建缺陷列表。并且必须确定 (利益相关者可以接受的) 解决缺陷的方法。

22.7.2 配置评审

确认过程的一个重要成分是配置评审。评审的目的是确保所有的软件配置元素已正确开发、编目, 且具有改善支持活动的必要细节。有时将配置评审称为审核 (audit), 这将在第 29 章详细讨论。

22.7.3 α 测试和 β 测试

对软件开发者而言, 预见用户如何实际使用程序几乎是不可能的。软件使用指南 (使用手册) 可能会被错误理解; 可能会使用令用户感到奇怪的数据组合; 测试者看起来很明显的输出对于工作现场的用户却是难以理解的。

为客户开发定制软件时, 执行一系列验收测试能使客户确认所有的需求。验收测试是由最终用户而不是软件工程师进行的, 它的范围从非正式的“测试驱动”直到有计划地、系统地进行一系列测试。实际上, 验收测试的执行可能超过几个星期甚至几个月, 因此, 可以发现长时间以来影响

引述 只要给予足够的关注, 所有 bug 都是容易找到的 (例如: 给予足够多的 β 测试人员和相关的开发人员, 几乎每个问题都能很快捕获, 且容易修改)。

E. Raymond

系统的累积错误。

若将软件开发为产品，由多个用户使用，让每个用户都进行正式的验收测试，这当然是不切实际的。多数软件开发者使用称为 α 测试与 β 测试的过程，以期查找到似乎只有最终用户才能发现的错误。

α 测试是由有代表性的最终用户在开发者的场所进行。软件在自然设置下使用，开发者站在用户的后面观看，并记录错误和使用问题。 α 测试在受控的环境下进行。

提问 α 测试和 β 测试之间的区别是什么？

β 测试在一个或多个最终用户场所进行。与 α 测试不同，开发者通常不在场，因此， β 测试是在不为开发者控制的环境下“现场”应用软件。最终用户记录测试过程中遇见的所有问题（现实存在的或想象的），并定期报告给开发者。接到 β 测试的问题报告之后，开发人员对软件进行修改，然后准备向最终用户发布软件产品。

β 测试的一种变体称为客户验收测试，有时是按照合同交付给客户时进行的。客户执行一系列的特定测试，试图在从开发者那里接收软件之前发现错误。在某些情况下（例如，大公司或政府系统），验收测试可能是非常正式的，会持续很多天甚至好几个星期。

SafeHome 准备确认

[场景] Doug Miller 的办公室，构件级设计及这些构件的构建工作正继续进行。

[人物] Doug Miller，软件工程经理；Vinod、Jamie、Ed 和 Shakira，SafeHome 软件工程团队成员。

[对话]

Doug：我们将在三个星期内准备好第一个增量的确认，怎么样？

Vinod：大概可以吧。集成进展得不错。我们每天执行冒烟测试，找到了一些 bug，但还没有我们处理不了的事情。到目前为止，一切都很好。

Doug：跟我谈谈确认。

Shakira：可以。我们将使用所有的用例作为测试设计的基础。目前我还没有开始，但我将为我负责的所有用例开发测试。

Ed：我这里也一样。

Jamie：我也一样。但是我们已经将确认测试与 α 测试和 β 测试一起考虑了，不是吗？

Doug：是，事实上我一直考虑请外包商帮我们做确认测试。在预算中我们有这笔钱……它将给我们新的思路。

Vinod：我认为确认测试已经在我们的控制之中了。

Doug：我确信是这样，但 ITG（独立测试组）能用另一种眼光来看这个软件。

Jamie：我们的时间很紧了，Doug，我没有时间培训新人来做这项工作。

Doug：我知道，我知道。但 ITG 仅根据需求和用例来工作，并不需要太多的培训。

Vinod：我仍然认为确认测试已经在我们的控制之中了。

Doug：我知道，Vinod，但在这方面我将强制执行。计划这周的后几天与 ITG 见面。让他们开始工作并看他们有什么意见。

Vinod：好的，或许这样做可以减轻工作负荷。

22.8 系统测试

在本书的开始,我们就强调过,软件只是基于计算机大系统的一部分。最终,软件要与其他系统成分(如硬件、人和信息)相结合,并执行一系列集成测试和确认测试。这些测试已超出软件过程的范围,而且不仅仅由软件工程师执行。然而,软件设计和测试期间所采取的步骤可以大大提高在大系统中成功地集成软件的可能性。

引述 与死亡和税收一样,测试既是令人不愉快的,也是不可避免的。

Ed Yourdon

一个传统的系统测试问题是“相互指责”。这种情况出现在发现错误时,每个系统成分的开发人员都因为这个问题抱怨别人。其实大家都不应该陷入这种无谓的争论之中,软件工程师应该预见潜在的接口问题,以及:(1)设计出错处理路径,用以测试来自系统其他成分的所有信息;(2)在软件接口处执行一系列模拟不良数据或其他潜在错误的测试;(3)记录测试结果,这些可作为出现“相互指责”时的“证据”;(4)参与系统测试的计划和设计,以保证软件得到充分的测试。

22.8.1 恢复测试

多数基于计算机的系统必须从错误中恢复并在一定的时间内重新运行。在有些情况下,系统必须是容错的,也就是说,处理错误绝不能使整个系统功能都停止。而在有些情况下,系统的错误必须在特定的时间内或严重的经济危害发生之前得到改正。

恢复测试是一种系统测试,通过各种方式强制地让系统发生故障,并验证其能适当恢复。若恢复是自动的(由系统自身完成),则对重新初始化、检查点机制、数据恢复和重新启动都要进行正确性评估。若恢复需要人工干预,则应计算平均恢复时间(Mean-Time-To-Repair, MTTR)以确定其值是否在可接受的范围之内。

22.8.2 安全测试

任何管理敏感信息或能够对个人造成不正当伤害(或带来好处)的计算机系统都是非礼或非法入侵的目标。入侵包括广泛的活动:黑客为了娱乐而试图入侵系统,不满的雇员为了报复而试图破坏系统,不良分子在非法利益驱使下试图入侵系统。

安全测试验证建立在系统内的保护机制是否能够实际保护系统不受非法入侵。引用Beizer[Bei84]的话来说:“系统的安全必须经受住正面的攻击,但是也必须能够经受住侧面和背后的攻击。”

只要有足够的时间和资源,好的安全测试最终将能够入侵系统。系统设计人员的作用是使攻破系统所付出的代价大于攻破系统之后获取信息的价值。安全测试和安全工程在第27章详细讨论。

22.8.3 压力测试

本章前面所讨论的软件测试步骤能够对正常的程序功能和性能进行彻底的评估。压力测试的目的是使软件面对非正常的情形。本质上,进行压力测试的测试人员会问:“在系统失效之前,能将系统的运行能力提高到什么程度?”

压力测试要求以一种非正常的数量、频率或容量的方式执行系统。例如:(1)在平均每秒出现1~2次中断的情形下,可以设计每秒产生10次中断的测试用例;(2)将输入数据

的量提高一个数量级以确定输入功能将如何反应；(3) 执行需要最大内存或其他资源的测试用例；(4) 设计可能在实际的运行系统中产生惨败的测试用例；(5) 创建可能会过多查找磁盘驻留数据的测试用例。从本质上来说，压力测试者将试图破坏程序。

压力测试的一个变体称为敏感性测试。在一些情况下（最常见的是在数学算法中），包含在有效数据界限之内的一小部分数据可能会引起极端处理情况，甚至是错误处理或性能的急剧下降。敏感性测试试图在有效输入类中发现可能会引发系统不稳定或者错误处理的数据组合。

引述 若你正在尽力查找实际系统的 bug，且没有为你的软件提供实际的压力测试，那么现在应该是你立即开始的时候了。

Boris Beizer

22.8.4 性能测试

对于实时和嵌入式系统，提供所需功能但不符合性能需求的软件是不能接受的。性能测试用来测试软件在集成环境中的运行性能。在测试过程的任何步骤中都可以进行性能测试。即使是在单元级，也可以在执行测试时评估单个模块的性能。然而，只有当整个系统的所有成分完全集成之后，才能确定系统的真实性能。

性能测试经常与压力测试一起进行，且常需要硬件和软件工具。也就是说，以严格的方式测量资源（例如处理器周期）的利用往往是必要的。当有运行间歇或事件（例如中断）发生时，外部工具可以监测到，并可定期监测采样机的状态。通过检测系统，测试人员可以发现导致效率降低和系统故障的情形。

22.8.5 部署测试

在很多情况下，软件必须在多种平台及操作系统环境中运行。有时也将部署测试称为配置测试，即在软件将要运行其中的每一种环境中测试软件。另外，部署测试检查客户将要使用的所有安装程序及专业安装软件（例如“安装程序”），并检查用于向最终用户介绍软件的所有文档。

487

软件工具 测试计划与管理

[目标] 这些工具辅助软件团队根据所选择的测试策略制订计划，并进行测试过程的管理。

[机制] 这类工具提供测试计划、测试存储、管理与控制、需求追踪、集成、错误追踪和报告生成。项目经理用这些工具作为项目策划工具的补充；测试人员利用这些工具计划测试活动，以及在测试进行时控制信息的流动。

[代表性工具]^①

• QaTraq 测试用例管理工具由 Traq Soft-

ware (www.testmanagement.com) 开发，“鼓励以结构化方法进行测试管理”。

- QAComplete 由 SmartBear (<http://SmartBear.com/products/qa-tools/test-management>) 开发，为管理敏捷测试过程的各个阶段提供单点控制。
- TestWorks 由 Software Research, Inc. (<http://www.testworks.com/>) 开发，包含一个完整的、集成的成套测试工具，包括测试管理和报告工具。
- OpensourceTesting.org (www.opensou-

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

rcetesting.org/testmgt.php) 列出了各种开源测试管理和计划工具。

- OpensourceManagement.com (http://

www.opensourcetestmanagement.com/) 列出了各种开源测试管理和计划工具。

22.9 调试技巧

软件测试是一种能够系统地加以计划和说明的过程，可以进行测试用例设计，定义测试策略，根据预期的结果评估测试结果。

调试 (debugging) 出现在成功的测试之后。也就是说，当测试用例发现错误时，调试是使错误消除的过程。尽管调试可以是也应该是一个有序的过程，但它仍然需要很多技巧。当评估测试结果时，软件工程师经常面对的是软件问题表现出的“症状”，即错误的外部表现与其内在原因之间可能并没有明显的关系。调试就是探究这一关系的智力过程。

22.9.1 调试过程

调试并不是测试，但总是发生在测试之后^①。参看图 22-7，执行测试用例，对测试结果进行评估，而且期望的表现与实际表现不一致时，调试过程就开始了。在很多情况下，这种不一致的数据是隐藏在背后的某种原因所表现出来的症状。调试试图找到隐藏在症状背后的原因，从而使错误得到修正。

引述 一旦我们开始编程，就会惊奇地发现，程序并不是像我们想象的那样容易正确。不得不去发现错误。我能记起那一刻，意识到从那时起我将花费大部分精力去查找自己程序中的错误。

Maurice Wilkes,
discovers
debugging, 1949

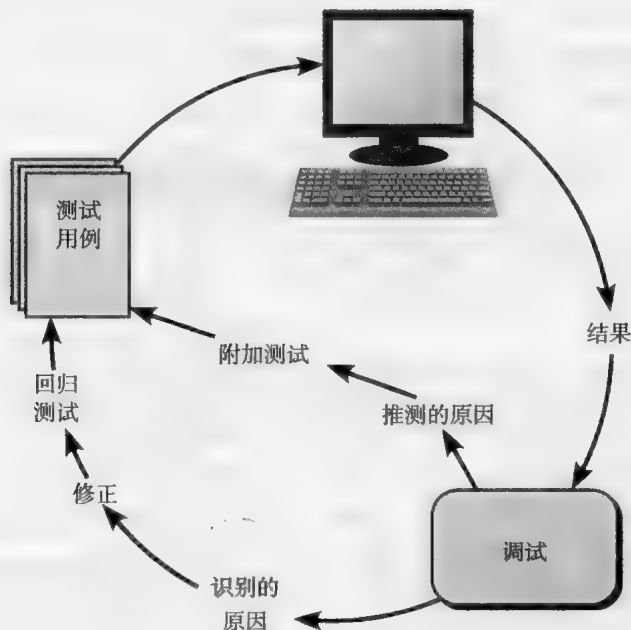


图 22-7 调试过程

^① 为了对此进行说明，我们采用最广泛的可能测试视图。在软件发布之前，不仅开发者要测试软件，客户/用户在每次使用软件之前也要对其进行测试。

调试过程通常得到以下两种结果之一：(1) 发现问题原因并将其改正；(2) 未能找到问题的原因。在后一种情况下，调试人员可以假设一个原因，设计一个或多个测试用例来帮助验证这个假设，重复此过程直到改正错误。

为什么调试如此困难？在很大程度上，人类心理（22.9.2 节）与这个问题的答案间的关系比软件技术更密切。然而，软件 bug 的以下特征为我们提供了一些线索。

1. 症状与原因出现的地方可能相隔很远。也就是说，症状可能在程序的一个地方出现，而原因实际上可能在很远的另一个地方。高度耦合的构件（第 12 章）加剧了这种情况的发生；
2. 症状可能在另一个错误被改正时（暂时）消失；
3. 症状实际上可能是由非错误因素（例如舍入误差）引起的；
4. 症状可能是由不易追踪的人为错误引起的；
5. 症状可能是由计时问题而不是处理问题引起的；
6. 重新产生完全一样的输入条件是困难的（如输入顺序不确定的实时应用）；
7. 症状可能时有时无，这在软硬件耦合的嵌入式系统中尤为常见；
8. 症状可能是由分布运行在不同处理器上的很多任务引起的。

在调试过程中，我们遇到错误的范围从恼人的小错误（如不正确的输出格式）到灾难性故障（如系统失效，造成严重的经济或物质损失）。错误越严重，查找错误原因的压力也就越大。通常情况下，这种压力会使软件开发人员在修改一个错误的同时引入两个甚至更多的错误。

22.9.2 心理因素

遗憾的是，有证据表明，调试本领属于一种个人天赋。一些人精于此道，而另一些人则不然。尽管有关调试的实验证据可以有多种解释，但对于具有相同教育和经验背景的程序员来说，他们的调试能力是有很大差别的。尽管学会调试可能比较困难，但仍然可以提出一些解决问题的方法。这些方法将在下一节讨论。

提问 为什么调试如此困难？

引述 每个人都
知道调试的难度
是首次写程序的
两倍。因此，如
果你像写它时一
样聪明，那么将
如何对它进行调
试呢？

Brian Kernighan

489

建议 设置一个
时限，比如两个
小时。在这个时
间限制内，尽力
独自调试程序。
然后，求助。

SafeHome 调试

[场景] Ed 的工作间，进行编码和单元测试。

[人物] Ed 和 Shakira，SafeHome 软件工程团队的成员。

[对话]

Shakira (经过工作间门口时向里张望): 嘿，午饭时你在哪儿？

Ed: 就在这里……工作。

Shakira: 你看上去很沮丧，怎么回事？

Ed (轻声地叹息): 我一直忙于解决这个

bug，从今天早晨 9:30 发现它之后，现在已下午 2:40 了，我还没有线索。

Shakira: 我想大家都同意在调试我们自己的东西时花费的时间不应该超过一小时，我们请求帮助，怎么样？

Ed: 好，但是……

Shakira (走进工作间): 什么问题？

Ed: 很复杂。而且，我查看这个问题已有 5 个小时，你不可能在 5 分钟内找到原因。

Shakira: 真让我兴奋，什么问题？

(Ed 向 Shakira 解释问题, Shakira 看了大约 30 秒没有说什么, 然后……)

Shakira (笑了): 哦, 就是那个地方, 在循环开始之前, 变量 setAlarmCondition

是不是不应该设置为 “false”?

(Ed 不相信地盯着屏幕, 向前躬着腰, 开始对着监视器轻轻地敲自己的头。Shakira 开怀大笑, 起身走出去。)

490

22.9.3 调试策略

不论使用什么方法, 调试有一个基本目标: 查找造成软件错误或缺陷的原因并改正。通过系统评估、直觉和运气相结合可以实现这个目标。

总的来说, 有三种调试方法 [Mye79]: 蛮干法、回溯法及原因排除法。这三种调试方法都可以手工执行, 但现代的调试工具可以使调试过程更有效。

调试方法。蛮干法可能是查找软件错误原因最常用但最低效的方法。在所有其他方法都失败的情况下, 我们才使用这种方法。利用“让计算机自己找错误”的思想, 进行内存转储, 实施运行时跟踪, 以及在程序中添加一些输出语句。希望在所产生的大量信息里可以让我们找到错误原因的线索。尽管产生的大量信息可能最终带来成功, 但更多的情况下, 这样做只是浪费精力和时间, 它将率先耗尽我们的想法!

引述 修改一个已坏的程序时, 第一步是让它重复失败(尽可能是在最简单的例子上)。

T. Duff

回溯法是比较常用的调试方法, 可以成功地应用于小程序中。从发现症状的地方开始, 向后追踪(手工)源代码, 直到发现错误的原因。遗憾的是, 随着源代码行数的增加, 潜在的回溯路径的数量可能会变得难以控制。

第三种调试方法——原因排除法——是通过演绎或归纳并引入二分法的概念来实现。对与错误出现相关的数据加以组织, 以分离出潜在的错误原因。假设一个错误原因, 利用前面提到的数据证明或反对这个假设。或者, 先列出所有可能的错误原因, 再执行测试逐个进行排除。若最初的测试显示出某个原因假设可能成立的话, 则要对数据进行细化以定位错误。

自动调试。以上调试方法都可以使用辅助调试工具。在尝试调试策略时, 调试工具为软件工程师提供半自动化的支持。Hailpern 与 Santhanam [Hai02] 总结这些工具的状况时写道: “人们已提出许多新的调试方法, 而且许多商业调试环境也已经具备。集成开发环境 (IDE) 提供了一种方法, 无需编译就可以捕捉特定语言的预置错误 (例如, 语句结束符的丢失、变量未定义等)。” 可用的工具包括各种调试编译器、动态调试辅助工具 (跟踪工具)、测试用例自动生成器和交互引用映射工具。然而, 工具不能替代基于完整设计模型和清晰源代码的仔细评估。

491

软件工具 调试

[目标] 这些工具为那些调试软件问题的人提供自动化的帮助, 目的是洞察那些用手
工调试可能难以捕捉的问题。

[机制] 大多数调试工具是针对特定编程语

言和环境的。

[代表性工具]^①

- Borland Silkt。由 Borland (<http://www.borland.com/products/>) 开发, 辅助测试

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

和调试。

- Coverty Development Testing Platform。由 Coverty (<http://www.coverty.com/products/>) 开发, 该工具将质量测试和安全性测试引入早期开发过程。
- C++Test。由 Parasoft (www.parasoft.com) 开发, 是一个单元测试工具, 对 C 和 C++ 代码的测试提供完全的支持。调试功能有助于已发现错误的诊断。
- CodeMedic。由 NewPlanet Software(www.newplanetsoftware.com/medic/) 开发,

为标准的 Unix 调试器 gdb 提供图形界面, 且实现了它的最重要特征。gdb 目前支持 C/C++、Java、PalmOS、各种嵌入式操作系统、汇编语言、FORTRAN 和 Modula-2。

- GNATS。一个免费应用软件 (www.gnu.org/software/gnats/), 是一组用于追踪 bug 报告的工具。

人为因素。若不提到强有力的助手——其他人, 那么有关调试方法和调试工具的任何讨论都是不完整的。有一个新颖的观点: 每个人都可能有为某个错误一直头痛的经历[⊖]。因此, 调试的最终箴言应该是: “若所有方法都失败了, 就该寻求帮助!”

22.9.4 纠正错误

一旦找到错误, 就必须纠正。但是, 我们已提到过, 修改一个错误可能会引入其他错误, 因此, 不当修改造成的危害会超过带来的益处。Van Vleck[Van89] 提出, 在进行消除错误原因的“修改”之前, 每个软件工程师应该问以下三个问题:

1. 这个错误的原因在程序的另一部分也产生过吗? 在多数情况下, 程序的错误是由错误的逻辑模式引起的, 这种逻辑模式可能会在别的地方出现。仔细考虑这种逻辑模式可能有助于发现其他错误。
2. 进行修改可能引发的“下一个错误”是什么? 在改正错误之前, 应该仔细考虑源代码(最好包括设计)以评估逻辑与数据结构之间的耦合。若要修改高度耦合的程序段, 则应格外小心。
3. 为避免这个错误, 我们首先应当做什么呢? 这个问题是建立统计软件质量保证方法的第一步(第 21 章)。若我们不仅修改了过程, 还修改了产品, 则不仅可以排除现在的程序错误, 还可以避免程序今后可能出现的错误。

引述 最好的测试人员不是发现错误最多的人, 而是纠正错误最多的人。

Cem Kaner et al.

492

22.10 小结

软件测试在软件过程中所占的技术工作量比例最大。不考虑所构建软件的类型, 系统测试计划、运行和控制策略从考虑软件的小元素开始, 逐渐面向整个软件。

软件测试的目标是发现错误。对于传统软件, 这个目标是通过一系列测试步骤达到的。单元测试和集成测试侧重于验证模块的功能以及将模块集成到程序结构中; 确认测试验证软件需求的可追溯性; 系统测试在软件集成为较大的系统时对软件进行确认。每个测试步骤都是通过有助于测试用例设计的一系列系统化测试技术来完成的。在每一步测试中, 用于考虑

⊖ 在设计软件和编码的过程中, 结对编程(第 5 章中所讨论的极限编程模型的一部分)提供了一种“排错”机制。

软件的抽象层次都得到了扩展。

面向对象软件的测试策略开始于类中操作的执行，然后转到以集成为目的的基于线程的测试。线程是响应输入或事件的一组类。基于使用的测试关注那些不与其他类过多协作的类。

对 WebApp 及移动 App 的测试方法与面向对象系统是一样的。然而，所设计的测试用于检查内容、功能性、界面、导航以及应用的性能和安全性方面。移动 App 需要特殊的测试方法，重点是在多种设备及实际网络环境中对应用程序进行测试。

与测试（测试是一种系统的、有计划的活动）不同的是，调试必须被看作一种技术。从问题的症状显示开始，调试活动要去追踪错误的原因。在调试过程可以利用的众多资源中，最有价值的是其他软件工程师的建议。

习题与思考题

493

- 22.1 用自己的话描述验证与确认的区别。两者都要使用测试用例设计方法和测试策略吗？
- 22.2 列出一些可能与独立测试组（ITG）的创建相关的问题。ITG 与 SQA 小组由相同的人员组成吗？
- 22.3 使用 22.1.3 节中描述的测试步骤来建立测试软件的策略总是可能的吗？对于嵌入式系统，会出现哪些可能的复杂情况？
- 22.4 为什么具有较高耦合度的模块难以进行单元测试？
- 22.5 “防错法”（antibugging，22.3.1 节）的概念是一个非常有效的方法。当发现错误时，它提供了内置调试帮助：
 - a. 为防错法开发一组指导原则。
 - b. 讨论利用这种技术的优点。
 - c. 讨论利用这种技术的缺点。
- 22.6 项目的进度安排是如何影响集成测试的？
- 22.7 在所有情况下，单元测试都是可能的或是值得做的吗？提供实例来说明你的理由。
- 22.8 谁应该完成确认测试——是软件开发人员还是软件使用者？说明你的理由。
- 22.9 为本书讨论的 SafeHome 系统开发一个完整的测试策略，并以测试规格说明的方式形成文档。
- 22.10 作为一个班级项目，为你的安装开发调试指南。这个指南应该提供面向语言和面向系统的建议。这些建议是通过总结学校学习过程中所遇到的挫折得到的。从一个经过全班和老师评审过的大纲开始，并在你的局部范围内将这个指南发布给其他人。

扩展阅读与信息资源

实际上，每本软件测试的书都讨论测试策略和测试用例设计方法。Everett 和 Raymond（《Software Testing》，Wiley-IEEE Computer Society Press, 2007）、Black（《Pragmatic Software Testing》，Wiley, 2007）、Spiller 和他的同事（《Software Testing Process: Test Management》，Rocky Nook, 2007）、Perry（《Effective methods for Software Testing》，3rd ed., Wiley, 2005）、Lewis（《Software Testing and Continuous Quality Improvement》，2nd ed., Auerbach, 2004）、Loveland 和他的同事（《Software Testing Techniques》，Charles River Media, 2004）、Burnstein（《Practical Software Testing Techniques》，Springer, 2003）、Dustin（《Effective Software Testing》，Addison-Wesley, 2002）以及 Kaner 和他的同事（《Lessons learned in Software Testing》，Wiley, 2001）所写的书只是讨论测试原理、概念、策略和方法的众多书籍中的一小部分。

对于敏捷软件开发方法有兴趣的读者，Gartner（《ATDD by Example: A Practical Guide to Acceptance Test-Driven Development》，Addison-Wesley, 2012）、Crispin 和 Gregory（《Agile Testing: A Practical Guide

for Testers and Teams》, Addison-Wesley, 2009)、Crispin 和 House (《Testing Extreme Programming》, Addison-Wesley, 2002) 以及 Beck (《Test Driven Development: By Example》, Addison-Wesley, 2002) 针对极限编程技术描述了测试策略与战术。Kamer 和他的同事 (《Lessons Learned in Software Testing》, Wiley, 2001) 描述了每个测试人员应该学习的 300 多条实用的“教训”(指导原则)。Watkins (《Testing IT: An Off-the Shelf Testing Process》(2nd ed.), Cambridge University Press, 2010) 为所有类型的软件(开发的和获取的)建立了有效的测试框架。Manages 和 O'Brien (《Agile Testing with Ruby and Rails》, Apress, 2008) 描述了针对 Ruby 编程语言和 Web 框架的测试策略和技术。

494

Bashir 和 Goel (《Testing Object-Oriented Software》, Springer-Verlag, 2012)、Sykes 和 McGregor (《Practical Guide to Testing Object-Oriented Software》, Addison-Wesley, 2001)、Binder (《Testing Object-Oriented Systems》, Addison-Wesley, 1999)、Kung 和他的同事 (《Testing Object-Oriented Software》, IEEE Computer Society Press, 1998) 以及 Marick (《The Craft of Software Testing》, Prentice-Hall, 1997) 描述了测试面向对象系统的策略与方法。

Grotker 和他的同事 (《The Developer's Guide to Debugging》(2nd ed.), CreateSpace Independent Publishing, 2012)、Whittaker (《Exploratory Testing》, Addison-Wesley, 2009)、Zeller (《Why Programs Fail: A Guide to Systematic Debugging》(2nd ed.), Morgan Kaufmann, 2009)、Butcher (《Debug It!》, Pragmatic Bookshelf, 2009)、Agans (《Debugging》, Amacon, 2006) 以及 Tells 和 Heieh (《The Science of Debugging》, The Coreolis Group, 2001) 所编写的书中包括调试指南。Kaspersky (《Hacker Debugging Uncovered》, A-list Publishing, 2005) 讲述了调试工具的技术。Younessi (《Object-Oriented Defect Management of Software》, Prentice-Hall, 2002) 描述了面向对象系统的缺陷管理技术。Beizer[Bei84] 描述了有趣的“bug 分类”, 这种分类引领了很多制定测试计划的有效方法。

Graham 和 Fewster (《Experience of Test Automation》, Addison-Wesley, 2012) 以及 Dustin 和他的同事 (《Implementing Automated Software Testing》, Addison-Wesley, 2009) 所编写的书讨论了自动测试。Hunt 和 John (《Java Performance》, Addison-Wesley, 2011)、Hewardt 和他的同事 (《Advanced .NET Debugging》, Addison-Wesley, 2009)、Matloff 和他的同事 (《The Art of Debugging with GDB, DDD, and Eclipse》, No Starch Press, 2008)、Madisetti 和 Akgul (《Debugging Embedded Systems》, Springer, 2007)、Robbins (《Debugging Microsoft .NET 2.0 Applications》, Microsoft Press, 2005)、Best (《Linux Debugging and Performance Tuning》, Prentice Hall, 2005)、Ford 和 Teorey (《Practical Debugging in C++》, Prentice Hall, 2002) 以及 Brown (《Debugging Perl》, McGraw-Hill, 2000)、Mitchell (《Debugging Java》, McGraw-Hill, 2000) 都针对书名所指的环境, 讲述了调试的特殊性质。

从网上可以获得大量有关软件测试策略的信息资源。与软件测试策略有关的最新的参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

495

测试传统的应用软件

要点浏览

概念：一旦生成了源代码，就必须对软件进行测试，以便在交付给客户之前尽可能多地发现（和改正）错误。我们的目标是设计一系列极有可能发现错误的测试用例。但是，如何做呢？这就是软件测试技术发挥作用的地方。这些技术为设计测试提供系统化的指导：（1）执行每个软件构件的内部逻辑和接口；（2）测试程序的输入和输出域以发现程序功能、行为和性能方面的错误。

人员：在测试的早期阶段，软件工程师完成所有的测试。然而，随着测试过程的进行，测试专家可能介入。

重要性：评审及其他软件质量保证活动可以且确实能够发现错误，但只有这些做法是远远不够的。每次执行程序时，用户都在测试它。因此，在程序交付给客户之前，就必须以发现并消除错误为目的来执行它。为了尽可能多地发现错误，

必须系统化地执行测试，而且必须利用严格的技术来设计测试用例。

步骤：对于传统的应用软件，可从两个不同的视角测试软件：（1）利用“白盒”测试用例设计技术执行程序内部逻辑；（2）利用“黑盒”测试用例设计技术确认软件需求。用例可辅助测试的设计，在软件确认的层面发现错误。在每种情况下，其基本意图都是以最少的工作量和最少的时间来发现最大数量的错误。

工作产品：设计一组测试用例使其不仅测试内部逻辑、接口、构件协作，还测试外部需求，并形成文档。定义期望结果，并记录实际结果。

质量保证措施：当开始测试时，改变视角，努力去“破坏”软件！规范化地设计测试用例，并对测试用例进行周密的评审。另外，评估测试覆盖率并追踪错误检测活动。

对于本质上具有建设性的软件工程师来说，测试展示出的是有趣的异常现象。测试要求开发者首先抛弃“刚开发的软件是正确的”这一先入为主的观念，然后努力去构造测试用例来“破坏”软件。Beizer[Bei90]有效地描述了这种情况：

有这样一个神话：若我们确实擅长编程，就应当不会有错误。只要我们确实很专注，只要每个人都使用结构化编程，采用自顶向下的设计方法……那么就不应该有错误。所以才有了这样的神话。神话中讲道：由于我们并不擅长所做的事，因此有错误存在。若不擅长，就应当感到内疚。因此，测试和测试用例的设计是对失败的承认，也是失败的一剂良药。测试的枯燥是对我们犯下的错误的处罚。为什么被罚？由于我们是人类？为

关键概念

基本路径测试
黑盒测试
边界值分析
控制结构测试
环路复杂性
等价类划分
流程图
图矩阵
基于图的测试方法

什么内疚？由于没能达到非人的完美境界？由于没能区分另一个程序员所想的和所说的之间存在的差异？由于没有心灵感应？由于没有解决交流问题？……由于人类四千年历史的缘故？

测试应该灌输内疚感吗？测试真的是摧毁性的吗？这些问题的回答是“不”！

本章针对传统的应用软件讨论软件测试用例设计技术。测试用例设计关注创建测试用例的一系列技术，这些测试用例的设计符合总体测试目标及第 22 章所述的测试策略。

23.1 软件测试基础

测试的目标是发现错误，并且好的测试发现错误的可能性较大。因此，软件工程师在设计与实现基于计算机的系统或产品时，应该想着可测试性。同时，测试本身必须展示一系列特征，达到以最少工作量发现最多错误的目标。

可测试性。James Bach^①为可测试性提供了下述定义：“软件可测试性就是（计算机程序）能够被测试的容易程度。”可测试的软件应具有下述特征。

可操作性。“运行得越好，越能有效地测试。”若设计和实现系统时具有质量意识，那么妨碍测试执行的错误将很少，从而使测试顺利进行。

可观察性。“你所看见的就是你所测试的。”作为测试的一部分所提供的输入会产生清楚的输出。测试执行期间系统状态和变量是可见的或可查询的，不正确的输出易于识别，内部错误会被自动检测和报告，源代码是可访问的。

可控制性。“对软件控制得越好，测试越能被自动执行和优化。”通过输入的某些组合可以产生所有可能的输出，并且输入/输出格式是一致的和结构化的。通过输入的组合，所有代码都可以执行到。测试工程师能够控制软硬件的状态和变量，能够方便地对测试进行说明、自动化执行和再现。

可分解性。“通过控制测试范围，能够更快地孤立问题，完成更灵巧的再测试。”软件由能够进行单独测试的独立模块组成。

简单性。“需要测试的内容越少，测试的速度越快。”程序应该展示功能简单性（例如，程序特性集是满足需求的最低要求）、结构简单性（例如，将体系结构模块化以限制错误的传播）以及代码简单性（例如，采用编码标准以使代码易于审查和维护）。

稳定性。“变更越少，对测试的破坏越小。”软件的变更不经常发生，变更发生时是可以控制的，且不影响已有的测试，软件失效后得到良好恢复。

易理解性。“得到的信息越多，进行的测试越灵巧。”体系结构设计以及内部构件、外部构件和共享构件之间的依赖关系能被较好地理解。技术文档可随时获取、组织合理、具体、详细且准确。设计的变更要通知测试人员。

可以使用 Batch 所建议的属性来开发易于测试的软件工作产品。

关键概念

基于模型的测试
正交数组测试
模式
白盒测试

引述 每个程序都做对某件事，可是那恰恰不一定是我们想让它做的事情！

作者不详

提问 可测试性的特征是什么？

497

引述 软件中的错误比其他技术中的错误更普遍、普遍且更烦人。

David Parnas

① 后面几段取得了 James Bach (copyright 1994) 的使用许可，并对最初出现在新闻组 comp.software-eng 的资料进行了改编。

测试特征。关于测试本身有哪些特征呢？Kaner、Falk 和 Nguyen[Kan93] 提出“好”的测试具有以下属性。

好的测试具有较高的发现错误的可能性。为达到这个目标，测试人员必须理解软件并尝试设想软件怎样才能失败。

提问 什么才是“好”的测试？

好的测试是不冗余的。测试时间和资源是有限的，执行与另一个测试有同样目标的测试是没有意义的。每个测试都应该有不同的目标（即使是细微的差别）。

好的测试应该是“最佳品种”[Kan93]。在一组具有类似目的的测试中，时间和资源的有限性会迫使只运行最有可能发现所有类别错误的测试。

好的测试应该既不太简单也不太复杂。尽管将一系列测试连接为一个测试用例有时是可能的，但潜在的副作用会掩盖错误。通常情况下，应该独立执行每个测试。

498

SafeHome 设计独特的测试

[场景] Vinod 的工作间。

[人物] Vinod 与 Ed, SafeHome 软件工程团队成员。

[对话]

Vinod: 这些是你打算用于测试操作 passwordValidation 的测试用例吗？

Ed: 是的，它们应该能覆盖用户进入时所有可能输入的密码。

Vinod: 让我看看……你提到正确的密码是 8080，对吗？

Ed: 嗯。

Vinod: 你指定密码 1234 和 6789 是要测试在识别无效密码方面的错误？

Ed: 对，我也测试与正确密码相接近的密

码，如 8081 和 8180。

Vinod: 那是可行的。但是我并不认为运行 1234 和 6789 两个输入有多大意义。这两个输入是冗余的……它们在测试同样的事情，不是吗？

Vinod: 确实是这样。倘若输入 1234 不能发现错误，换句话说，操作 passwordValidation 指出它是无效密码，那么输入 6789 也不可能显示任何新的东西。

Ed: 我明白你的意思。

Vinod: 我不是吹毛求疵，只是我们做测试的时间有限，因此，好的方法是运行最有可能发现新错误的测试。

Ed: 没问题……我再想想。

23.2 测试的内部视角和外部视角

任何工程化的产品（以及大多数其他东西）都可以采用以下两种方式之一进行测试：（1）了解已设计的产品要完成的指定功能，可以执行测试以显示每个功能是可操作的，同时，查找在每个功能中的错误；（2）了解产品的内部工作情况，可以执行测试以确保“所有的齿轮吻合”——即内部操作依据规格说明执行，而且对所有的内部构件已进行了充分测试。第一种测试方法采用外部视角，也称为黑盒测试；第二种方法采用内部视角，也称为白盒测试。^①

黑盒测试暗指在软件接口处执行测试。黑盒测试检查系统的功能方面，而不考虑软件的内部结构。软件的白盒测试是基于过程细节的封闭检查。通过提供检

引述 在设计测试用例中只有一条规则，那就是覆盖所有特征，但并不创建太多的测试用例。

Tsuneo Yamaura

① 术语功能测试和结构测试有时分别用于代替黑盒测试和白盒测试。

查特定条件集或循环的测试用例，测试将贯穿软件的逻辑路径和构件间的协作。

乍一看，好像是全面的白盒测试将获得“100% 正确的程序”。我们需要做的只是识别所有的逻辑路径，开发相应的测试用、例执行测试用例并评估结果；即生成测试用例，彻底地测试程序逻辑。遗憾的是，穷举测试存在某种逻辑问题，即使对于小程序，可能的逻辑路径的数量也可能非常大。然而，不应该觉得白盒测试不切实际而抛弃这种方法。可以选择并测试有限数量的重要逻辑路径，检测重要数据结构的有效性。

关键点 只有在构件级设计（或源代码）存在之后，才设计白盒测试。此时，一定要获得程序的逻辑细节。

499

信息栏 穷举测试

考虑 100 行的 C 语言程序。一些基本的数据声明之后，程序包含两个嵌套循环，依靠输入指定的条件，每个从 1 到 20 次进行循环。在内部循环中，需要 4 个 if-then-else 结构。这个程序中大约有 1014 个可能的执行路径！

为了说明这个数字代表的含义，我们假设已经开发了一个神奇的测试处理器（“神奇”意味着没有这样的处理器存在）

来做穷举测试。在 1 毫秒内，处理器可以开发一个测试用例、执行测试用例并评估测试结果。若处理器每天工作 24 小时，每年工作 365 天，要对这个程序做完穷举测试，需要工作 3170 年。不可否认，这将对大多数的开发进度造成巨大障碍。

因此，可以肯定地说，对于大型软件系统，穷举测试是不可能的。

23.3 白盒测试

白盒测试有时也称为玻璃盒测试或结构化测试，是一种测试用例设计方法，它利用作为构件级设计的一部分所描述的控制结构来生成测试用例。利用白盒测试方法导出的测试用例可以：（1）保证一个模块中的所有独立路径至少被执行一次；（2）对所有的逻辑判定均需测试取真（true）和取假（false）两个方面；（3）在上下边界及可操作的范围内执行所有循环；（4）检验内部数据结构以确保其有效性。

引述 bug 潜在角落并在边界处聚集。

Boris Beizer

23.4 基本路径测试

基本路径测试是由 Tom McCabe[McC76] 首先提出的一种白盒测试技术。基本路径测试方法允许测试用例设计者计算出过程设计的逻辑复杂性测量，并以这种测量为指导来定义执行路径的基本集。执行该基本集导出的测试用例保证程序中的每一条语句至少执行一次。

23.4.1 流程图表示

在介绍基本路径方法之前，必须介绍一种简单的控制流表示方法，称为流程图（或程序图）^①。流程图利用图 23-1 所示的表示描述逻辑控制流。每种结构化构造（第 14 章）都有相应

500

① 事实上，不使用流程图也可以执行基本路径测试方法，但是，流程图是用于理解控制流和解释方法的一种有用表示。

的流图符号。

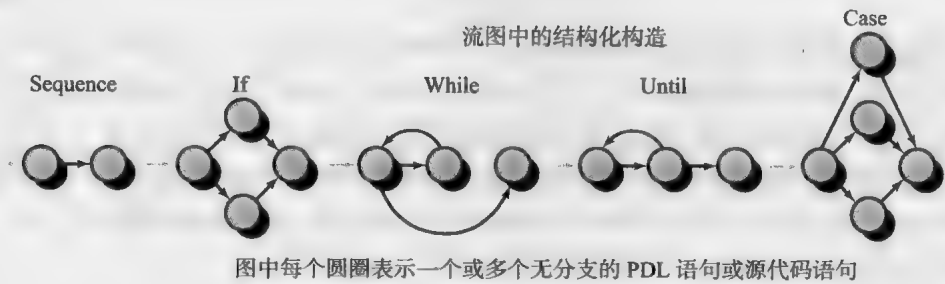


图 23-1 流图表示

为了说明流图的使用，考虑图 23-2a 所示的过程设计表示。这里，流程图用于描述程序的控制结构。图 23-2b 将这个流程图映射为相应的流图（假设流程图的菱形判定框中不包含复合条件）。在图 23-2b 中，圆称为流图结点（flow graph node），表示一个或多个过程语句。处理框序列和一个菱形判定框可以映射为单个结点。流图中的箭头称为边或连接，表示控制流，类似于流程图中的箭头。一条边必须终止于一个结点，即使该结点并不代表任何过程语句（例如表示 if-then-else 结构的流图符号）。由边和结点限定的区域称为域。计算域时，将图的外部作为一个域^①。

建议 仅在构件的逻辑结构复杂的情况下，才应该画流图。使用流图可以更轻易地追踪程序路径。

501

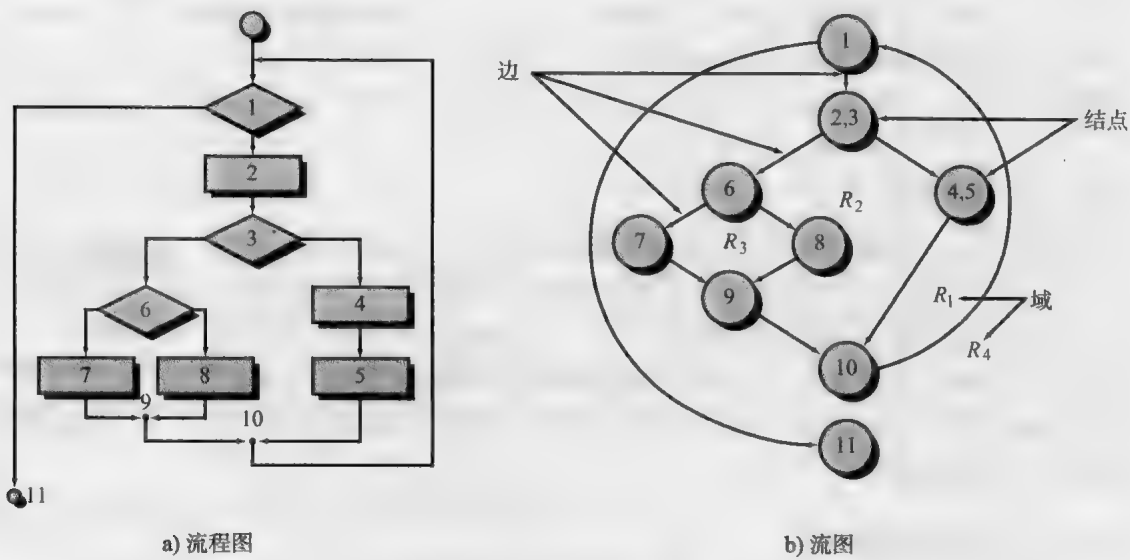


图 23-2 流程图和流图

在过程设计中遇到复合条件时，流图的生成会变得稍微复杂一些。当一个条件语句中存在一个或多个布尔运算符（逻辑 OR、AND、NAND、NOR）时，复合条件就出现了。图 23-3 给出了一段程序设计语言（PDL）程序及其对应的流图。注意，分别为条件语句“IF a OR b”的每个条件（a 和 b）创建不同的结点。包含条件的结点称为判定结点，其特征是由它发射出两条或多条边。

① 23.6.1 节将更详细地讨论图及其使用。

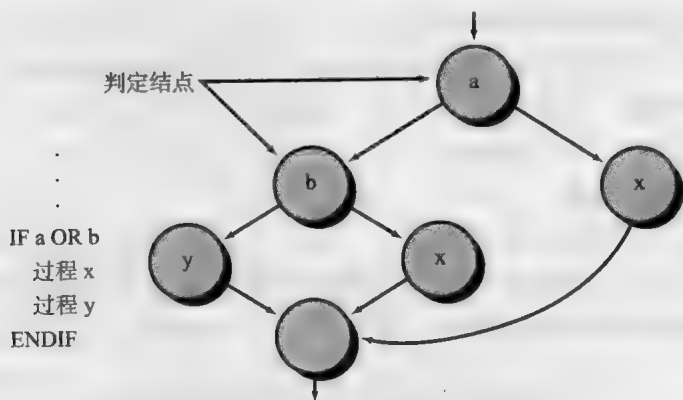


图 23-3 复合逻辑

23.4.2 独立程序路径

独立路径是任何贯穿程序的、至少引入一组新处理语句或一个新条件的路径。当按照流程图进行描述时，独立路径必须沿着至少一条边移动。这条边在定义该路径之前未被遍历。例如，图 23-2b 所示流图的一组独立路径如下：

路径 1: 1-11

路径 2: 1-2-3-4-5-10-1-11

路径 3: 1-2-3-6-8-9-10-1-11

路径 4: 1-2-3-6-7-9-10-1-11

注意，每条新的路径引入一条新边，路径

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

不是一条独立路径，因为它不过是已提到路径的简单连接，而没有引入任何新边。

路径 1、2、3 和 4 构成图 23-2b 所示流图的基本集合。也就是说，若设计测试以强迫执行这些路径（基本集合），则可以保证程序中的每条语句至少执行一次，且每个条件的取真和取假都被执行。应该注意到，基本集合不是唯一的。事实上，对给定的过程设计，可以导出很多不同的基本集合。

如何知道要找出多少路径？环复杂性的计算提供了答案。环复杂性是一种软件度量，它为程序的逻辑复杂度提供了一个量化的测度。用在基本路径测试方法的环境下时，环复杂性的值定义了程序基本集合中的独立路径数，并提供了保证所有语句至少执行一次所需测试数量的上限。

环复杂性以图论为基础，并提供了非常有用的软件度量。可以通过以下三种方法之一来计算环复杂性。

1. 流图中域的数量与环复杂性相对应。

2. 对于流图 G ，环复杂性 $V(G)$ 定义如下：

$$V(G) = E - N + 2$$

其中 E 为流图的边数， N 为流图的结点数。

3. 对于流图 G ，环复杂性 $V(G)$ 也可以定义如下：

$$V(G) = P + 1$$

建议 在预见易于出错的模块方面，环复杂性是一种有用的度量，可以用于做测试计划以及测试用例设计。

提问 如何计算环复杂性？

其中 P 为包含在流图 G 中的判定结点数。

再回到图 23-2b 中的流图，环复杂性可以通过上述三种算法来计算。

1. 该流图有 4 个域。

2. $V(G) = 11$ (边数) $- 9$ (结点数) $+ 2 = 4$ 。

3. $V(G) = 3$ (判定结点数) $+ 1 = 4$ 。

因此，图 23-2b 中流图的环复杂性是 4。

更重要的是， $V(G)$ 的值提供了组成基本集合的独立路径的上界，并由此得出覆盖所有程序语句所需设计和运行的测试数量的上界。

关键点 环复杂性提供保证程序中每条语句至少执行一次所需测试用例数的上界。

503

SafeHome

使用环复杂性

[场景] Shakira 的工作间。

[人物] Vinod 和 Shakira, SafeHome 软件工程团队成员，他们正在为安全功能准备测试计划。

[对话]

Shakira：看，我知道应该对安全功能的所有构件进行单元测试，但是，如果考虑所有必须测试的操作的数量，工作量就太大了，我不知道……可能我们应该放弃白盒测试，将所有的构件集成在一起，开始执行黑盒测试。

Vinod：你估计我们没有足够的时间做构件测试、检查操作，然后集成，是不是？

Shakira：第一次增量测试的最后期限离我们很近了……是的，我有点担心。

Vinod：你为什么不对最有可能出错的操作执行白盒测试呢？

Shakira (愤怒地)：我怎么能够准确地知

道哪个是最易出错的呢？

Vinod：环复杂性。

Shakira：嗯？

Vinod：环复杂性。只要计算每个构件中每个操作的环复杂性。看看哪些操作的 $V(G)$ 具有最高值。那些操作就是最有可能出错的操作。

Shakira：怎么计算 $V(G)$ 呢？

Vinod：那相当容易。这里有本书说明了怎么计算。

Shakira (翻看那几页)：好了，这计算看上去并不难。我试一试。具有最高 $V(G)$ 值的就是要做白盒测试的候选操作。

Vinod：但还要记住，这并不是绝对的，那些 $V(G)$ 值低的构件还是可能有错的。

Shakira：好吧。但这至少降低了必须进行白盒测试的构件数。

23.4.3 生成测试用例

基本路径测试方法可以应用于过程设计或源代码。在本节中，我们将基本路径测试描述为一系列步骤。以图 23-4 中用 PDL 描述的过程 average 为例，说明测试用例设计方法中的各个步骤。注意，尽管过程 average 是一个非常简单的算法，但却包含了复合条件与循环。下列步骤可用于生成基本测试用例集。

引述 犯错误的是人，发现错误的是神。

Robert Dunn

1. 以设计或源代码为基础，画出相应的流图。利用 23.4.1 节给出的符号和构造规则创建流图。参见图 23-4 中过程 average 的 PDL 描述，将那些 PDL 语句进行编号，并映射到相应的流图结点，以此来创建流图。图 23-5 给出了相应的流图。

2. 确定所得流图的环境复杂性。通过运用 23.4.2 节描述的算法来确定环境复杂性 $V(G)$ 的值。应该注意到, 不建立流图也可以确定 $V(G)$, 方法是通过计算 PDL 中条件语句的数量 (过程 average 的复合条件语句计数为 2), 然后加 1。在图 23-5 中:

$$V(G) = 6(\text{域数})$$

$$V(G) = 17(\text{边数}) - 13(\text{结点数}) + 2 = 6$$

$$V(G) = 5(\text{判定结点数}) + 1 = 6$$

3. 确定线性独立路径的基本集合。 $V(G)$ 的值提供了程序控制结构中线性独立路径的数量。在过程 average 中, 我们指定了 6 条路径:

路径 1: 1-2-10-11-13

路径 2: 1-2-10-12-13

路径 3: 1-2-3-10-11-13

路径 4: 1-2-3-4-5-8-9-2-...

路径 5: 1-2-3-4-5-6-8-9-2-...

路径 6: 1-2-3-4-5-6-7-8-9-2-...

路径 4、5、6 后面的省略号 (...) 表示可加上控制结构其余部分的任意路径。在设计测试用例的过程中, 经常通过识别判定结点作为导出测试用例的辅助手段。本例中, 结点 2、3、5、6 和 10 为判定结点。

PROCEDURE average;

* This procedure computes the average of 100 or fewer numbers that lie between bounding values; it also computes the sum and the total number valid.

INTERFACE RETURNS average, total, input, total, valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total, input, total, valid;
minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;

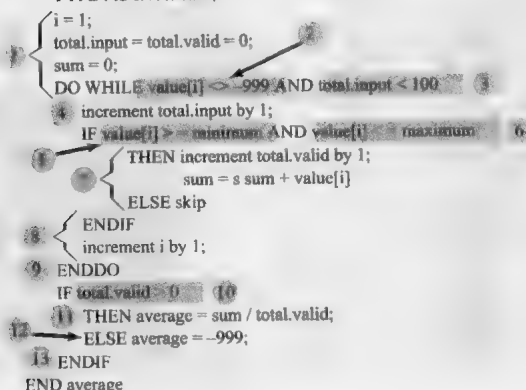


图 23-4 已标识结点的 PDL

4. 准备测试用例, 强制执行基本集合中的每条路径。测试人员应该选择测试数据, 以便在测试每条路径时适当地设置判定结点的条件。执行每个测试用例并将结果与期望值进行比较。一旦完成了所有的测试用例, 测试人员就可以确信程序中所有的语句至少已被执行一次。

引述 只是由于在将 64 位浮点值转换为 16 位整数的操作中包含了一个软件缺陷 (代码错误), Ariane 5 型火箭在升空时发生爆炸。这枚火箭和它的 4 颗卫星都没有投保, 它们价值 5 亿美元。如果进行路径测试是可以发现这个错误的, 但由于预算原因被否決了。
——一篇新闻报道

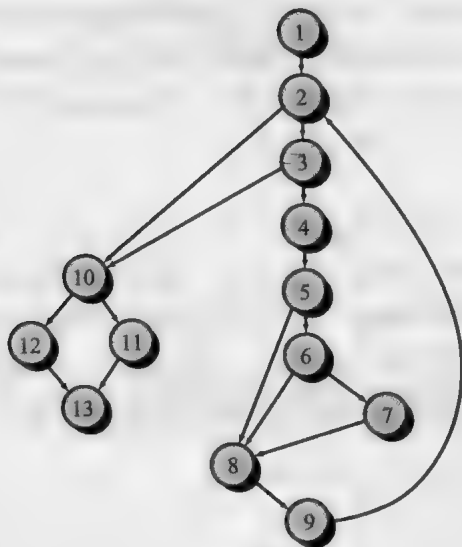


图 23-5 过程 average 的流图

注意，某些独立路径（本例中的路径 1）不能单独进行测试。也就是说，遍历路径所需的数据组合不能形成程序的正常流。在这种情况下，将这些路径作为另一个路径的一部分进行测试。

23.4.4 图矩阵

导出流图甚至确定基本路径集合的过程都可以机械化。一种称为图矩阵（graph matrix）的数据结构对于开发辅助基本路径测试的软件工具相当有用。

图矩阵是一种方阵，其大小（即行与列的数量）等于流图的结点数。每行和每列都对应于已标识的结点，矩阵中的项对应于结点间的连接（边）。图 23-6 给出了一个简单流图及相应的图矩阵 [Bei90]。

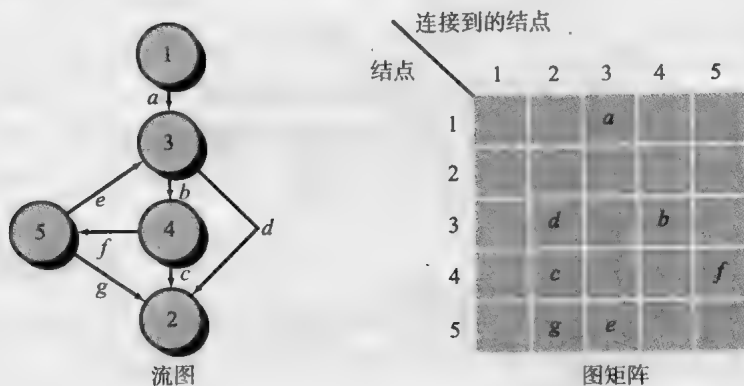


图 23-6 图矩阵

如图 23-6 所示，流图的每个结点用数字标识，而每条边用字母标识。矩阵中的每个字母对应于纵横方向两结点间的连接，例如，边 *b* 连接结点 3 和结点 4。

从这种意义上讲，图矩阵只是流图的表格表示。然而，通过为每个矩

提问 什么是图矩阵？如何对其进行扩展以用于测试？

阵项加入一个连接权值，图矩阵将成为测试期间评估程序控制结构的一个强有力的工具。连接权值提供了有关控制流的附加信息。最简单的情况下，连接权值是 1（连接存在）或 0（连接不存在），但是，可以赋予连接权值其他更有意义的特征：

- 执行连接（边）的概率。
- 遍历连接的处理时间。
- 遍历连接时所需要的内存。
- 遍历连接时所需要的资源。

Beizer[Bei90] 提供了可用于图矩阵的其他数学算法的全面讨论。利用这些技术，设计测试用例时所需进行的分析可以部分或完全自动化。

23.5 控制结构测试

23.4 节所描述的基本路径测试是控制结构测试技术之一。虽然基本路径测试简单且高效，但其本身并不充分。本节简单讨论控制结构测试的其他变体，这些技术拓宽了测试的覆盖率并提高了白盒测试的质量。

条件测试 [Tai89] 通过检查程序模块中包含的逻辑条件进行测试用例设计。数据流测试 [Fra93] 根据程序中变量的定义和使用位置来选择程序的测试路径。

循环测试是一种白盒测试技术，完全侧重于循环构建的有效性。可以定义 4 种不同的循环 [Bei90]：简单循环、串接循环、嵌套循环和非结构化循环（图 23-7）。

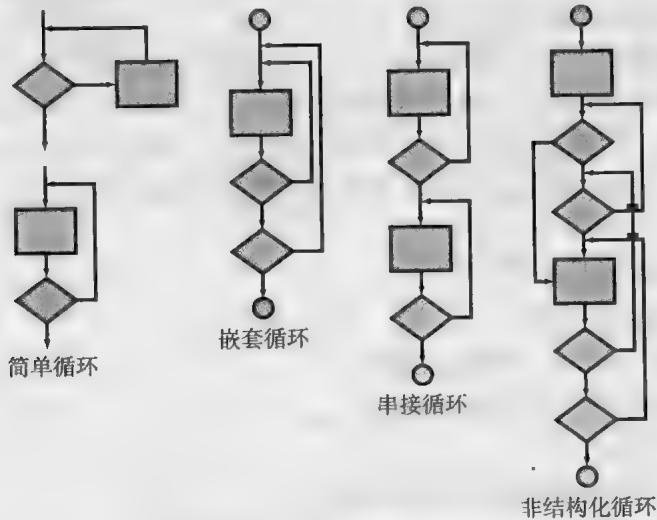


图 23-7 循环的类别

简单循环。下列测试集可用于简单循环，其中， n 是允许通过循环的最大次数。

1. 跳过整个循环。
2. 只有一次通过循环。
3. 两次通过循环。
4. m 次通过循环，其中 $m < n$ 。
5. $n-1$ 、 n 、 $n+1$ 次通过循环。

引述 优秀的测试人员是注意到“奇怪的事情”就会对它采取行动的大师。

Brian Marick

嵌套循环。若将简单循环的测试方法扩展应用于嵌套循环，则可能的测试数将随着嵌套层次的增加而成几何级数增长。这将导致不切实际的测试数量。Beizer[Bei90]提出了一种有助于减少测试数的方法。

1. 从最内层循环开始，将其他循环设置为最小值。
2. 对最内层循环执行简单循环测试，而使外层循环的迭代参数（例如循环计数）值最小，并对范围以外或不包括在内的值增加其他测试。
3. 由内向外构造下一个循环的测试，但使其他外层循环具有最小值，并使其他嵌套循环为“典型”值。
4. 继续上述过程，直到测试完所有的循环。

串接循环。若串接循环的每个循环彼此独立，则可以使用简单循环测试方法。然而，若两个循环串接起来，且第一个循环的循环计数为第二个循环的初始值，则这两个循环并不独立。若循环不独立，则建议使用嵌套循环的测试方法。

建议 不能对非结构化循环进行有效测试，需要对它们进行重新设计。

非结构化循环。若有可能，应该重新设计这类循环以反映结构化程序结构的使用（第14章）。

23.6 黑盒测试

黑盒测试也称行为测试或功能测试，侧重于软件的功能需求。黑盒测试使软件工程师能设计出可以测试程序所有功能需求的输入条件集。黑盒测试并不是白盒测试的替代品，而是作为发现其他类型错误的辅助方法。

黑盒测试试图发现以下类型的错误：（1）不正确或遗漏的功能；（2）接口错误；（3）数据结构或外部数据库访问错误；（4）行为或性能错误；（5）初始化和终止错误。

与白盒测试不同，白盒测试在测试过程的早期执行，而黑盒测试倾向于应用在测试的后期阶段（第22章）。黑盒测试故意不考虑控制结构，而是侧重于信息域。设计黑盒测试要回答下述问题：

- 如何测试功能的有效性？
- 如何测试系统的行为和性能？
- 哪种类型的输入会产生好的测试用例？
- 系统是否对特定的输入值特别敏感？
- 如何分离数据类的边界？
- 系统能承受什么样的数据速率和数据量？
- 特定类型的数据组合会对系统运行产生什么样的影响？

提问 什么是黑盒测试必须回答的问题？

通过运用黑盒测试技术，可以生成满足下述准则的测试用例集 [Myc79]：能够减少达到合理测试所需的附加测试用例数，并且能够告知某些错误类型是否存在，而不是仅仅知道与特定测试相关的错误。

23.6.1 基于图的测试方法

黑盒测试的第一步是理解软件中建模的对象^①及这些对象间的关系。这一步一旦完成，

① 这里，我们在最广泛的环境中考虑术语“对象”。它包括数据对象、传统的构件（模块）以及计算机软件的面向对象元素。

下一步就是定义一系列验证“所有对象之间具有预期关系”的测试 [Bei95]。换言之，软件测试首先是创建重要对象及其关系图，然后设计覆盖图的一系列测试用例，使得图中的每个对象和关系都测试到，并发现错误。

为完成这些步骤，软件工程师首先要创建图，其中结点表示对象，连接表示对象间的关系，结点权值描述结点的属性（例如，具体的数据值或状态行为），连接权值描述连接的某些特征。

图的符号表示如图 23-8a 所示。结点用圆表示。而连接有几种形式，有向连接（用箭头表示）表示这种关系只在一个方向存在，双向连接（也称对称连接）表示关系适用于两个方向，并行连接表示图结点间有几种不同的关系。

关键点 图表示数据对象与程序对象间的关系，它使我们能够设计测试用例，查找与这些关系有关的错误。

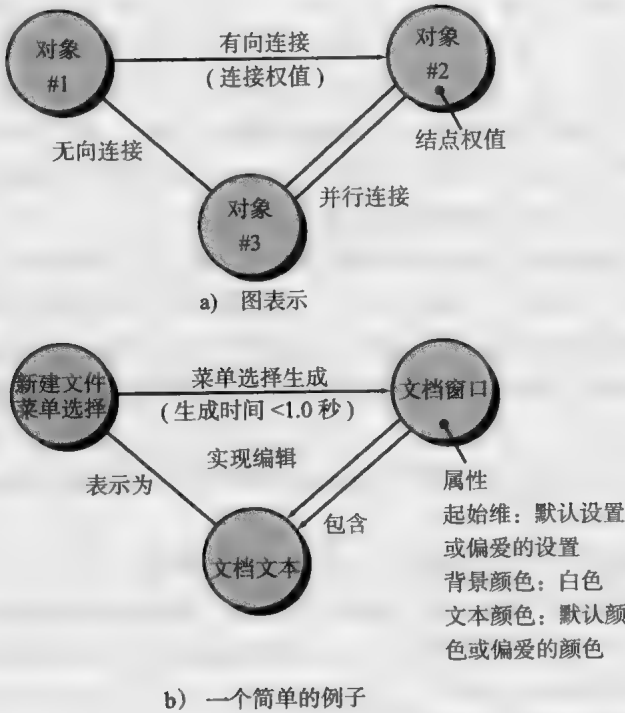


图 23-8 图符号表示及示例

考虑一个简单的例子，字处理应用中图的一部分，如图 23-8b 所示，其中

- 对象 #1 = 新建文件（菜单选择）
- 对象 #2 = 文档窗口
- 对象 #3 = 文档文本

该图中，选择菜单“新建文件”生成一个“文档窗口”。“文档窗口”的结点权值提供窗口生成时预期的属性集。连接权值表明必须在 1.0 秒之内生成。一条无向连接在“新建文件”菜单选择和“文档文本”之间建立对称关系。并行连接显示“文档窗口”与“文档文本”间的关系。事实上，设计测试用例还需要更详细的图描述。然后软件工程师通过遍历图并覆盖图中所示的关系来设计测试用例。这些测试用例用于发现各种关系中的错误。Beizer[Bei95]描述了下面几种使用图的行为测试方法。

事务流建模。结点表示事务的步骤（例如，利用联机服务预订机票所需的步骤）。连接

表示这些步骤间的逻辑连接。例如，数据对象“航班信息输入”的后面跟着“确认有效性的处理”操作。

有限状态建模。结点表示用户可见的不同软件状态（例如，订票人员处理电话订票时的各个屏幕），连接表示状态间的转换（例如，在“库存有效性检查”期间，“订单信息”会得到验证，之后会输入“客户账单信息”）。状态图（第 11 章）可用于辅助创建这种图。

数据流建模。结点表示数据对象，而连接为一个数据对象转换为其他数据对象时发生的变换。例如，结点扣缴税款（FTW）由总工资（GW）利用关系 $FTW = 0.62 \times GW$ 计算出来。

时间建模。结点为程序对象，连接是对象间的顺序连接。连接权值用于指定程序执行时所需的执行时间。

基于图的测试方法的详细讨论超出了本书的范围。感兴趣的读者参看 [Bei95]，可以对其有全面的了解。

23.6.2 等价类划分

[511]

等价类划分是一种黑盒测试方法，它将程序的输入划分为若干个数据类，从中生成测试用例。理想的测试用例可以单独发现一类错误（例如，所有字符数据处理不正确），否则在观察到一般的错误之前需要运行许多测试用例。

等价类划分的测试用例设计是基于对输入条件的等价类进行评估。利用上节引入的概念，若对象可以由具有对称性、传递性和自反性的关系连接，则存在等价类 [Bei95]。等价类表示输入条件的一组有效的或无效的状态。通常情况下，输入条件要么是一个特定值、一个数据域、一组相关的值，要么是一个布尔条件。可以根据下述指导原则定义等价类。

提问 如何为测试定义等价类？

1. 若输入条件指定一个范围，则可以定义一个有效等价类和两个无效等价类。
2. 若输入条件需要特定的值，则可以定义一个有效等价类和两个无效等价类。
3. 若输入条件指定集合的某个元素，则可以定义一个有效等价类和一个无效等价类。
4. 若输入条件为布尔值，则可以定义一个有效等价类和一个无效等价类。

通过运用设计等价类的指导原则，可以为每个输入域数据对象设计测试用例并执行。选择测试用例以便一次测试一个等价类的尽可能多的属性。

23.6.3 边界值分析

大量错误发生在输入域的边界处，而不是发生在输入域的“中间”。这是将边界值分析（Boundary Value Analysis, BVA）作为一种测试技术的原因。边界值分析选择一组测试用例检查边界值。

边界值分析是一种测试用例设计技术，是对“等价划分”的补充。BVA 不是选择等价类的任何元素，而是在等价类“边缘”上选择测试用例。BVA 不是仅仅侧重于输入条件，它也从输出域中导出测试用例 [Mye79]。

BVA 的指导原则在很多方面类似于等价划分的原则。

1. 若输入条件指定为以 a 和 b 为边界的范围，则测试用例应该包括 a 和 b ，略大于和略小于 a 和 b 。
2. 若输入条件指定为一组值，则测试用例应当执行其中的最大值和最

引述 测试代码的一种有效方式是在其自然边界处运行它。

Brian Kernighan

关键点 通过侧重考虑一个等价类“边界”处的数据，BVA 扩展了等价类划分。

小值，以及略大于和略小于最大值和最小值的值。

3. 指导原则 1 和 2 也适用于输出条件。例如，工程分析程序要求输出温度和压强的对照表，应该设计测试用例创建输出报告，输出报告可生成所允许的最大（和最小）数目的表项。

512

4. 若内部程序数据结构有预定义的边界值（例如，表具有 100 项的定义限制），则一定要设计测试用例，在其边界处测试数据结构。

大多数软件工程师会在某种程度上凭直觉完成 BVA。通过运用这些指导原则，边界测试会更加完全，从而更有可能发现错误。

23.6.4 正交数组测试

许多应用程序的输入域是相对有限的。也就是说，输入参数的数量不多，且每个参数可取的值有明确的界定。当这些数量非常小时（例如，3 个输入参数，取值分别为 3 个离散值），则有可能考虑每个输入排列，并对所有的输入域进行测试。然而，随着输入值数量的增加及每个数据项的离散值数量的增加，穷举测试将是不切实际或不可能的。

关键点 正交数组测试使得软件工程师设计的测试用例能够以合理的数量提供最大的测试覆盖。

正交数组测试（orthogonal array testing）可以应用于输入域相对较小但对穷举测试而言又过大的问题。正交数组测试方法对于发现区域错误（region fault）（有关软件构件内部错误逻辑的一类错误）尤其有效。

为说明正交数组测试与更传统的“一次一个输入项”方法之间的区别，考虑有 3 个输入项 X 、 Y 和 Z 的系统。每个输入项有 3 个不同的离散值。这样可能有 $3^3 = 27$ 个测试用例。Phadke[Pha97] 提出了一种几何观点，用于组织与 X 、 Y 和 Z 相关的测试用例。如图 23-9 所示，一个输入项一次可能沿着某个输入轴在顺序上有变化。这导致了相对有限的输入域覆盖率（图 23-9 中左图立方体所示）。

513

使用正交数组测试时，创建测试用例的一个 L9 正交数组。L9 正交数组具有“平衡特性” [Pha97]，即测试用例（图中表示为黑点）“均匀地分散在整个测试域中”，如图 23-9 中右图立方体所示。这样整个输入域的测试覆盖会更完全。

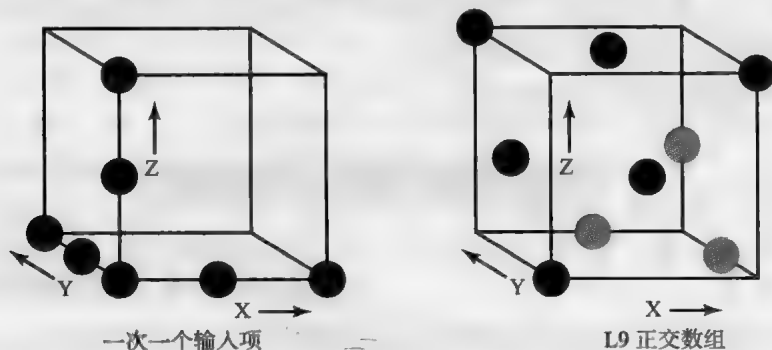


图 23-9 测试用例的几何视图 [Pha97]

为了说明 L9 正交数组的使用，考虑传真应用中的 send 函数。向函数 send 传递 4 个参数 P1、P2、P3 和 P4。其中每个参数取 3 个不同的值。例如，P1 的取值：

P1= 1，现在发送

P1= 2，一小时后发送

P1=3, 半夜 12 点后发送

P2、P3 和 P4 也分别取值 1、2 和 3, 表示其他发送功能。

如果选择“一次一个输入项”的测试策略, 则测试 (P1, P2, P3, P4) 的测试序列如下: (1,1,1,1), (2,1,1,1), (3,1,1,1), (1,2,1,1), (1,3,1,1), (1,1,2,1), (1,1,3,1), (1,1,1,2), (1,1,1,3)。但是这些测试数据只会揭示单模式错误 [Pha97], 也就是说, 这些错误由一个参数触发。

给定相对少量的输入参数和离散值, 穷举测试是可能的。所需要的测试数为 $3^4 = 81$, 虽比较大, 但还是能够做到的。可以发现所有与数据项排列相关的错误, 但所需的工作量较大。

正交数组测试方法使我们可以提供较好的测试覆盖, 而测试用例比穷举测试少得多。send 函数的 L9 正交数组如图 23-10 所示。

Phadke [Pha97] 对利用 L9 正交数组测试方法的测试结果评价如下。

检测和分离所有单模式错误。单模式错误是任意单个参数在任意级别上的一致性问题的。例如, 若因子 $P1 = 1$ 的所有测试用例产生一个错误条件, 则它就是一个单模式错误。在这个例子中, 测试 1、2 和 3 (图 23-10) 将显示错误。通过分析哪些测试显示了错误的信息, 可以识别出是哪个参数值产生了错误。在这个例子中, 注意到测试 1、2 和 3 产生错误, 因而可以将其分离 (有关“现在发送 ($P1 = 1$)”的逻辑处理) 为错误源。这样的错误分离对于修改错误是很重要的。

检测所有双模式错误。若当两个参数的特定级别一起出现时存在一致性问题, 则称之为双模式错误。实际上, 双模式错误表示成对不相容问题或两个测试参数间的有害干扰问题。

多模式错误。(所显示类型的) 正交数组仅可以保证单模式和双模式错误的检测。然而, 有些多模式错误也可以通过这些测试检测出来。

正交数组测试的详细讨论见 [Pha89]。

测试用例	测试参数			
	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

图 23-10 L9 正交数组

软件工具 测试用例设计

[目标] 辅助软件团队设计完整的黑盒测试和白盒测试用例集。

[机制] 这些工具可分为静态测试与动态测试两大类。在产业界, 有三种不同类型的静态测试工具: 基于代码的测试工具、基于专用测试语言的测试工具和基于需求的测试工具。基于代码的测试工具所接收的输入为源代码, 完成一系列分析,

最后生成测试用例。专用测试语言 (例如 ATLAS) 使软件工程师能够书写详细的测试规格说明, 在规格说明中, 描述每个测试用例及其执行逻辑。基于需求的测试工具将特定用户需求进行分离, 为检查需求的测试用例 (或测试类) 的设计提出建议。动态测试工具与执行程序进行交互, 检查路径覆盖, 测试特定变量值的断言, 或监

测程序的执行流。

[代表性工具]^①

- McCabe Test。由 McCabe & Associates (www.mccabe.com) 开发, 它实现了一些从环复杂性评估和其他软件度量中派生的路径测试技术。
- TestWorks。由 Software Research (<http://www.testworks.com/stwhome.html>) 开发, 它是一套完整的自动化测试工具, 有助于开发 C、C++ 和 Java 软件的测试

用例设计, 并为回归测试提供支持。

- T-VEC Test Generation System。由 T-VEC Technologies (www.t-vec.com) 开发, 它是支持单元测试、集成测试和确认测试的工具集。通过使用面向对象需求规格说明中的信息辅助测试用例的设计。
- e-Test Suite。由 Empirix (www.empirix.com) 开发, 拥有测试 WebApp 的完整工具集, 包括辅助测试用例设计工具和测试计划工具。

515

23.7 基于模型的测试

基于模型的测试 (Model-based Testing, MBT) 是一种黑盒测试技术, 它使用需求模型中的信息作为生成测试用例的基础 [DAC03]。在很多情况下, 基于模型的测试技术使用 UML 状态图——一种行为模型 (第 11 章) 作为测试用例设计的基础^②。MBT 技术需要以下 5 个步骤。

引述 当你在代码中寻找错误时, 很难发现它; 当你认为自己的代码没有错误时, 就更难发现它。

Steve McConnell

1. 分析软件的已有行为模型或创建一个行为模型。回忆一下, 行为模型指明软件是如何响应外部事件或刺激的。为了创建行为模型, 我们需要执行第 11 章所讨论的步骤: (1) 评价所有的用例, 以完全理解系统内的交互顺序; (2) 标识驱动交互顺序的事件, 并理解这些事件如何与特定的对象相关; (3) 为每个用例创建交互顺序; (4) 构造系统的 UML 状态图 (例如, 见图 11-1); (5) 评审行为模型, 验证其精确性和一致性。
2. 遍历行为模型, 并标明促使软件在状态之间进行转换的输入。输入将触发事件, 使转换发生。
3. 评估行为模型, 并标注当软件在状态之间转换时所期望的输出。回想一下, 每个转换都由一个事件触发, 作为转换的结果, 某些方法会被调用并产生输出。对于步骤 2 所指定的每个输入 (用例) 集合, 指定所期望的输出, 以说明它们在行为模型中的特点。
4. 运行测试用例。可以手工执行测试, 也可以创建测试脚本并使用测试工具执行测试。
5. 比较实际结果和期望结果, 并根据需要进行调整。

MBT 可帮助我们发现软件行为中的错误, 因此, 它在测试事件驱动的应用时也非常有用。

23.8 文档测试和帮助设施测试

软件测试一词造成一种假象: 大量的测试用例是为检查计算机程序和它们所管理的数据做准备的。但是, 帮助设施或文档中的错误与数据或源代码中的错误一样, 它们都会影响程序的验收。完全按照用户指南或在线帮助进行操作, 但得到的结果或行为却与文档的描述不

516

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

② 当软件需求是用决策表、语法或 Markov 链表示时, 也可以使用基于模型的测试 [DAC03]。

符，没有什么比这种情况更让人沮丧了。因此，文档测试应该是所有软件测试计划中有意义的一部分。

文档测试可分为两个阶段进行。第一阶段为技术评审（第 20 章），检查文档编辑的清晰性；第二阶段是现场测试（live test），结合实际程序使用文档。

令人意外的是，对文档的现场测试竟可以采用与前面讨论的许多黑盒测试方法相似的技术，包括：基于图的测试可用于描述程序的使用；等价类划分和边界值分析方法可用于定义各种输入类和相关的交互操作；MBT 则可用于确保文档规定的行为和实际行为的吻合。因而程序的用法可以贯穿全部文档而得到追踪。

信息栏 文档测试

在测试文档和帮助设施时，应该回答下列问题：

- 该文档准确地描述了如何完成每种使用模式吗？
- 每种交互序列的描述是否准确？
- 实例准确吗？
- 术语、菜单描述以及系统响应与实际程序一致吗？
- 在文档中能够比较容易地得到指导吗？
- 利用该文档可以容易地完成疑难解答吗？
- 该文档的目录和索引是否健壮、准确和完整？

- 该文档的设计（布局、字体、缩进、图表）有助于信息的理解与快速吸收吗？
- 所有显示给用户的软件错误信息在该文档中有更详细的描述吗？对看到错误信息后所采取的行动有明确的描述吗？
- 如果提供超文本链接，链接是否准确和完整？
- 如果提供超文本链接，导航设计是否适合信息获取？

回答这些问题唯一可行的方法是让独立的第三方（如选定的用户）在程序使用的环境下测试该文档。应该记录所有的差异，确定模糊或薄弱的地方，以方便可能的重写。

23.9 实时系统的测试

许多实时应用的时间依赖性和异步特征给测试带来了新的困难——时间。测试用例设计者不仅必须考虑传统的测试用例，而且要考虑事件处理（即中断处理）、数据的定时以及处理数据的任务（进程）的并行性。在许多情况下，实时系统在一种状态下提供的测试数据可以正常处理，而在另一种状态下提供同样的数据将会出现错误。

例如，控制复印机的实时软件在机器处于复印状态时，接收操作员的中断（即机器操作员按控制键，如 RESET 或 DARKEN）不会产生错误。若同一操作员中断出现在机器处于卡纸状态时，则会显示诊断代码，指明卡纸的位置将丢失（一个错误）。

此外，实时系统的软件和硬件环境之间的密切关系也会导致测试问题。软件测试必须考虑硬件故障对软件处理的影响。这种故障很难实时模拟。

对于实时软件的测试，可以提出以下 4 个步骤的策略。

任务测试。单独测试每个任务。也就是说，对每个任务设计并执行传统的测试。在测试期间，每个任务单独执行。任务测试可以发现逻辑和功能错误，但不能发现时间或行为错误。

提问 测试实时系统的有效策略是什么？

行为测试。利用通过自动化工具创建的系统模型，是可以模拟实时系统的行为并按照外部事件序列检查其行为的。这些分析活动可以作为测试用例设计的基础。当实时软件建成时，执行这些测试用例。使用类似等价划分的技术（23.6.2 节），对事件（如中断、控制信号）进行分类测试。例如，复印机的事件可能是用户中断（如重置计数器）、机械中断（如卡纸）、系统中断（如碳粉低）及失效模式（如滚筒过热）。这些事件的每一个都要单独测试，并检查可执行系统的行为，以检测与这些事件有关的处理错误。

任务间测试。一旦单个任务和系统行为中的错误已经分离出来，测试就要转向与时间相关的错误。用不同的数据速率和处理负载来测试任务间的异步通信，以确定任务间是否发生同步错误。另外，通过消息队列和数据存储进行通信任务的测试，以发现这些数据存储区域大小方面的错误。

系统测试。集成软件与硬件并进行全范围的系统测试以发现软件/硬件接口处的错误。多数实时系统都能处理中断，因此，测试布尔事件的处理尤其重要。利用状态图（第 11 章），测试人员开发所有可能的中断和中断处理列表，然后设计测试以评估下列系统特征：

- 是否正确赋予和处理中断优先级？
- 每个中断的处理是否正确？
- 中断处理过程的性能（如处理时间）是否符合需求？
- 关键时刻若出现大量中断，是否会导致功能和性能上的问题？

另外，作为中断处理的一部分并用于传输信息的全局数据区域也应该测试，以评估产生副作用的可能性。

23.10 软件测试模式

模式作为描述特定设计问题解决方案的一种机制，其使用已经在第 16 章讨论过了。但模式也可以用于提出其他软件工程解决方案——此处为软件测试。测试模式描述常见的测试问题和解决方案，可以辅助软件工程师处理这些问题。

在过去 10 年间，大多数软件测试已是一项专门的活动。如果测试模式有助于软件测试团队对软件测试进行更有效的交流、理解采用特定测试方法的动机，以及将测试用例的设计作为一种进化活动，以使每次迭代都产生更完整的测试用例，那么测试模式就达到了预期的目的。

测试模式可以采用与设计模式（第 16 章）同样的方式进行描述。文献（例如 [BIN99]、[Mar02]）中已提出了几十种测试模式。下面的三种测试模式（仅以摘要的形式给出）是较有代表性的例子。

模式名称：结对测试

摘要：一种面向过程的模式，结对测试描述了一种与结对编程（第 5 章）类似的技术。在这种测试模式中，两个测试人员一起设计并执行一系列测试，可以应用于单元测试、集成测试或确认测试活动中。

模式名称：独立测试接口

摘要：在面向对象系统中需要对每个类进行测试，包括“内部类”（不向使用它们的外部构件暴露任何接口的类）。独立测试接口模式描述如何创建“一个测试接口，该测试接口可用于描述一些类（这些类仅对某个内部构件可见）的特定测试”[Lan01]。

网络资源 软件测试模式目录可在 <http://c2.com/cgi-bin/> 找到。

关键点 测试模式有助于软件团队对测试进行更有效的交流，并对采用特定测试方法的影响力有更好的理解。

518

519

模式名称：场景测试

摘要：一旦已经执行了单元测试与集成测试，就需要确定软件是否能够以让用户满意的方式执行。场景测试描述一种从用户的角度测试软件的技术。在这个层次上的失败表明软件不能满足用户的可见需求 [Kan01]。

对测试模式的全面讨论超出了本书的范围。对于这个重要主题的其他信息，有兴趣的读者可以参看 [Bin99]、[Mar02] 和 [Tho04]。

23.11 小结

测试用例设计的主要目标是设计最有可能发现软件错误的测试用例集。为达到这个目标，可采用两种不同的测试用例设计技术：白盒测试和黑盒测试。

白盒测试侧重于程序控制结构。设计测试用例以保证测试期间程序中所有的语句至少被执行一次，且所有的逻辑条件都得到检查。基本路径测试是一种白盒测试技术，利用程序图（或图矩阵）生成保证覆盖率的线性无关的测试集。条件和数据流测试进一步检查程序逻辑，循环测试作为白盒测试技术的补充，检查不同复杂度的循环。

Hetzel[Het84] 将白盒测试描述为“小型测试”。他的意思是，本章所考虑的白盒测试一般应用于小的程序构件（例如模块或一小组模块）。而黑盒测试放宽了测试的焦点，可以将其称为“大型测试”。

黑盒测试用来确认功能需求，而不考虑程序的内部结构。黑盒测试技术侧重于软件的信息域，通过划分程序的输入域和输出域来设计测试用例，以提供完全的测试覆盖。等价划分将输入域划分为有可能检查软件特定功能的数据类。边界值分析则检查程序在可接受的限度内处理边界数据的能力。正交数组测试提供了一种高效的、系统的、使用少量的输入参数的测试方法。基于模型的测试使用需求模型的元素测试应用的行为。

520

有经验的软件开发人员经常说：“测试永无止境，它只不过是从小软件工程师转移到用户。客户每次使用程序时都是一次测试。”通过运用测试用例设计，软件工程师可以取得更完全的测试，因此可以在“客户的测试”开始之前，发现和改正尽可能多的错误。

习题与思考题

- 23.1 Myers[Mye79] 用以下程序作为对测试能力的自我评估：某程序读入 3 个整数值，这 3 个整数值表示三角形的 3 条边。该程序打印信息以表明三角形是不规则的、等腰的或等边的。开发一组测试用例测试该程序。
- 23.2 设计并实现习题 23.1 描述的程序（适当时使用错误处理）。从该程序中导出流程图并用基本路径测试方法设计测试，以保证程序中的所有语句都被测试到。执行测试用例并显示结果。
- 23.3 你能够想出 23.1.1 节中没有讨论的其他测试目标吗？
- 23.4 选择一个你最近设计和实现的构件。设计一组测试用例，保证利用基本路径测试执行所有的语句。
- 23.5 说明、设计和实现一个软件工具，使其能够对你所选的程序设计语言计算环复杂性。在你的设计中，利用图矩阵作为有效的数据结构。
- 23.6 阅读 Beizer[Bei95] 或相关的网络资源（例如，www.laynetworks.com/Discrete%20Mathematics_1g.htm），并确定如何扩展习题 23.5 所开发的程序以适应各种连接权值。扩展你的工具以处理执行概率或连接处理时间。
- 23.7 设计一个自动化测试工具，使其能够识别循环并按照 23.5.3 节中的方法分类。
- 23.8 扩展习题 23.7 中描述的工具，为曾经遇到的每个循环类生成测试用例。与测试人员交互地完成

这个功能是有必要的。

23.9 至少给出 3 个例子, 在这些例子中, 黑盒测试能够给人“一切正常”的印象, 而白盒测试可能发现错误。再至少给出 3 个例子, 在这些例子中白盒测试可能给人“一切正常”的印象, 而黑盒测试可能发现错误。

23.10 穷举测试(即便对非常小的程序)是否能够保证程序 100% 正确?

23.11 测试你经常使用的某个应用软件的用户手册(或帮助设施)。在文档中至少找到一个错误。

扩展阅读与信息资源

实际上, 所有软件测试方面的书籍都同时考虑测试策略和测试技术。因此, 第 22 章的推荐读物同样适用于本章。有许多讨论测试原理、概念、策略和方法的书籍, 下面的书籍只是其中的一小部分: Burnstein (《 Practical Software Testing 》, Springer, 2010)、Crispin 和 Gregory (《 Agile Testing: A Practical Guide for Testers and Agile Teams 》, Addison-Wesley, 2009)、Lewis (《 Software Testing and Continuous Quality Improvement 》, 3rd ed., Auerbach, 2008)、Ammann 和 Offutt (《 Introduction to Software Testing 》, Cambridge University Press, 2008)、Everett 和 McCleod (《 Software Testing 》, Wiley-IEEE Computer Society Press, 2007)、Black (《 Pragmatic Software Testing 》, Wiley, 2007)、Spiller 和他的同事 (《 Software Testing Process : Test Management 》, Rocky Nook, 2007)、Perry (《 Effective Methods for Software Testing 》, 3rd ed., Wiley, 2006)、Loveland 和他的同事 (《 Software Testing Techniques 》, Charles River Media, 2004)、Dustin (《 Effective Software Testing 》, Addison-Wesley, 2002)、Craig 和 Kaskiel (《 Systematic Software Testing 》, Artech House, 2002)、Tamres (《 Introducing Software Testing 》, Addison-Wesley, 2002) 以及 Whittaker (《 Exploratory Software Testing: Tips, Tricks, and Techniques to Guide Test Design 》, Addison-Wesley, 2009 和《 How to Break Software 》, Addison-Wesley, 2002)。

Myers[Mye79] 经典书籍的第 3 版 (《 The Art of Software Testing, 3rd ed., Wiley, 2011 》) 由 Myers 及其同事编写, 非常详细地讲述了测试用例的设计技术。Black (《 Managing the Testing Process 》, 3rd ed., Wiley, 2009)、Jorgensen (《 Software Testing: A Craftsman's Approach 》, 3rd ed., CRC Press, 2008)、Pezze 和 Young (《 Software Testing and Analysis 》, Wiley, 2007)、Perry (《 Effective Methods for Software Testing 》, 3rd ed., Wiley, 2006)、Copeland (《 A Practitioner's Guide to Software Test Design 》, Artech, 2003) 以及 Hutcheson (《 Software Testing Fundamentals 》, Wiley, 2003) 都提供了测试用例设计方法和技术的有用介绍。Beizer[Bei90] 的经典文本全面介绍了白盒测试技术, 引入了数学级别上的严格性, 这在其他测试方面的论述中一般是不具备的。他后来的书籍 [Bei95] 对重要方法作了简明介绍。

软件测试是一种资源密集型的活动。为此, 许多组织为部分测试过程提供了自动化支持。Graham 和她的同事 (《 Experiences of Test Automation: Case Studies of Software Test Automation 》, Addison-Wesley, 2012 和《 Software Test Automation 》, Addison-Wesley, 1999)、Li 和 Wu (《 Effective Software Test Automation 》, Sybex, 2004)、Mosely 和 Posey (《 Just Enough Software Test Automation 》, Prentice Hall, 2002)、Poston (《 Automating Specification-Based Software Testing 》, IEEE Computer Society, 1996) 以及 Dustin、Rashka 和 Poston (《 Automated Software Testing: Introduction, Management, and Performance 》, Addison-Wesley, 1999) 讨论了自动化测试的工具、策略和方法。Ngyuen 和他的同事 (《 Happy About Global Software Test Automation 》, Happy About Press, 2006) 介绍了测试自动化的执行视图。

Meszaros (《 Unit Test Patterns: Refactoring Test Code 》, Addison-Wesley, 2007)、Thomas 和他的同事 (《 Java Testing Patterns 》, Wiley, 2004) 以及 Binder[Bin99] 描述了测试模式, 包括方法测试、类/簇测试、子系统测试、可复用构件测试、框架测试、系统测试、测试自动化及特定数据库测试。

从网上可以获得大量的有关测试用例设计方法的信息。有关测试技术的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

521

522

测试面向对象的应用

要点浏览

概念: 面向对象 (OO) 软件的体系结构是包含协作类的一系列分层的子系统。这些系统的每个元素 (子系统和类) 所执行的功能都有助于满足系统需求。有必要在各种不同的层次上测试面向对象系统, 尽力发现当类之间存在协作以及子系统穿越体系结构层通信时可能发生的错误。

人员: 面向对象测试由软件工程师和测试专家执行。

重要性: 在将程序交付给客户之前, 必须运行程序, 试图去除所有的错误, 使得客户免受糟糕软件产品的折磨。为了发现尽可能多的错误, 必须进行系统的测试, 并且必须使用严格的技术来设计测试用例。

步骤: 面向对象测试在策略上类似传统系统的测试, 但在战术上是不同的。面向对象分析和设计模型在结构和内容上类似于

最终的面向对象程序, 因此“测试”开始于对这些模型的评审。一旦代码已经生成, 面向对象测试就开始进行“小规模”的类测试。设计一系列测试以检查类操作及一个类与其他类协作时是否存在错误。当将类集成起来构成子系统时, 应用基于线程的测试、基于使用的测试、簇测试以及基于故障的测试方法彻底检查协作类。最后, 运用用例 (作为分析模型的一部分开发) 发现软件确认级的错误。

工作产品: 设计并文档化一组测试用例来检查类、协作和行为。定义期望的结果, 并记录实际结果。

质量保证措施: 测试时应改变观点, 努力去“破坏”软件! 规范化地设计测试用例, 并对测试用例进行周密的评审。

在第 23 章已经提到, 简单地说, 测试的目标就是在可行的时间期限内, 以可行的工作量发现最大可能数量的错误。虽然这个基本目标对于面向对象软件没有改变, 但面向对象程序的本质改变了测试策略和测试战术。

可以断定, 由于可复用类库规模的增大, 更多的复用会缓解对面向对象系统进行繁重测试的需求。然而确切地说, 相反的情况的确也是存在的。Binder[Bin94b] 在讨论这种情况时说道:

每次复用都是一种新的使用环境, 重新测试需要谨慎。为了在面向对象系统中获得高可靠性, 似乎需要更多的测试, 而不是更少的测试。

为了充分测试面向对象的系统, 必须做三件事情: (1) 对测试的定义进行扩展, 使其包括应用于面向对象分析和设计模型的错误发现技术; (2) 单元测试和集成测试策略必须彻底改变; (3) 测试用例设计必须考虑面向对象软件的独特性质。

关键概念

- 类测试
- 簇测试
- 一致性
- 基于故障的测试
- 多类测试
- 面向对象模型
- 划分测试
- 随机测试
- 基于场景的测试
- 测试方法
- 测试策略
- 基于线程的测试
- 基于使用的测试

24.1 扩展测试的视野

面向对象软件的构造开始于分析和设计模型的创建^①。由于面向对象软件工程模式的进化特性,这些模型开始于系统需求的不太正式的表示,并进化到更详细的类模型、类关系、系统设计和分配以及对象设计(通过消息传递来合并对象连接模型)。在每一个阶段,都要对模型进行“测试”,尽量在错误传播到下一轮迭代之前发现错误。

可以肯定,面向对象分析和设计模型的评审非常有用,因为相同的语义结构(例如,类、属性、操作、消息)出现在分析、设计和代码层次。因此,在分析期间所发现的类属性的定义问题会防止副作用的发生。如果问题直到设计或编码阶段(或者是分析的下一轮迭代)还没有发现,副作用就会发生。

例如,在分析的第一轮迭代中,考虑定义了很多属性的一个类。有一个无关的属性被扩展到类中(由于对问题域的错误理解),然后指定了两个操作来处理此属性。对分析模型进行了评审,领域专家指出了这个问题。在这个阶段去除无关的属性,可以在分析阶段避免下面的问题和不必要的工作量。

1. 可能会生成特殊的子类,以适应不必要的属性或例外。去除无关的属性后,与创建不必要的子类相关的工作就可以避免。

2. 类定义的错误解释可能导致不正确或多余的类关系。

3. 为了适应无关的属性,系统的行为或类可能被赋予不适当的特性。

如果问题没有在设计期间被发现以致于进一步传播,则在设计期间会发生以下问题(早期的评审可以避免这些问题的发生)。

1. 在系统设计期间,可能会发生将类错误地分配给子系统和任务的情况。

2. 可能会扩展不必要的设计工作,比如为涉及无关属性的操作创建过程设计。

3. 消息模型可能不正确(因为会为无关的操作设计消息)。

如果问题没有在设计期间检测出来,以致于传递到编码活动中,那么将大幅增加生成代码的工作量,用于实现不必要的属性、两个不必要的操作、驱动对象间通信的消息以及很多其他相关的问题。另外,类的测试会消耗更多不必要的时间。一旦最终发现了这个问题,一定要对系统执行修改,以处理由变更所引起的潜在副作用。

在开发的后期,面向对象分析(OOA)和面向对象设计(OOD)模型提供了有关系统结构和行为的实质性信息。因此,在代码生成之前,需要对这些模型进行严格的评审。

应该在模型的语法、语义和语用方面对所有的面向对象模型进行正确性、完整性和一致性测试(在这里,术语测试包括技术评审)。

24.2 测试 OOA 和 OOD 模型

不能在传统意义上对分析和设计模型进行测试,因为这些模型是不能运行的。然而,可以使用技术评审(第20章)检查模型的正确性和一致性。

建议 尽管面向对象分析和设计模型的评审是测试面向对象应用不可分割的一部分,但要认识到这是不够的,还要实施可运行的测试。

524

引述 我们使用的工具对我们的思考习惯具有深远的影响,因此,也对我们的思考能力具有深远的影响。

Edsger Dijkstra

① 分析建模和设计建模技术在本书第二部分介绍。基本的面向对象概念在附录2中介绍。

24.2.1 OOA 和 OOD 模型的正确性

用于表示分析和设计模型的符号和语法是与为项目所选择的特定分析和设计方法连接在一起的。由于语法的正确性是基于符号表示的正确使用来判断的，因此必须对每个模型进行评审以确保维持了正确的建模习惯。

在分析和设计期间，可以根据模型是否符合真实世界的问题域来评估模型的语义正确性。如果模型准确地反映了现实世界（详细程度与模型被评审的开发阶段相适应），则在语义上是正确的。实际上，为了确定模型是否反映了现实世界的需求，应该将其介绍给问题领域的专家，由专家检查类定义以及层次中遗漏和不清楚的地方。要对类关系（实例连接）进行评估，确定这些关系是否准确地反映了现实世界的对象连接[⊖]。

24.2.2 面向对象模型的一致性

面向对象模型的一致性可以通过这样的方法来判断：“考虑模型中实体之间的关系。不一致的分析模型或设计模型在某一部分中的表示没有正确地反映到模型的其他部分” [McG94]。

为了评估一致性，应该检查每个类及其与其他类的连接。可以使用类－职责－协作者（Class-Responsibility-Collaborator, CRC）模型和对象－关系图来辅助此活动。如在第 10 章所学到的，CRC 模型由 CRC 索引卡片组成。每张 CRC 卡片都列出了类的名称、职责（操作）和协作者（接收其消息的其他类及完成其职责所依赖的其他类）。协作意味着面向对象系统的类之间的一系列关系（即连接）。对象关系模型提供了类之间连接的图形表示。这些信息都可以从分析模型（第 10 章）中获得。

推荐使用下面的步骤对类模型进行评估 [McG94]。

- 1. 检查 CRC 模型和对象－关系模型。对这两个模型做交叉检查，确保需求模型所蕴含的所有协作都已正确地反映在这两个模型中。
- 2. 检查每一张 CRC 索引卡片的描述以确定委托职责是协作者定义的一部分。例如，考虑为销售积分结账系统定义的类（称为 CreditSale），这个类的 CRC 索引卡片如图 24-1 所示。

类的名称: credit sale	
类的类型: transaction event	
类的特性: nontangible, atornio, esquential, pemanert, guarded	
职责:	协作者
读信用卡	信用卡
取得授权	信用权利
显示购物金额	产品票
	销售总账
	审计文件
生成账单	账单

图 24-1 用于评审的 CRC 索引卡片实例

⊖ 对于面向对象系统，在对照现实世界的使用场景追踪分析和设计模型方面，用例是非常有价值的。

对于这组类和协作,例如,将职责(例如读信用卡)委托给已命名的协作者(CreditCard),看看此协作者是否完成了这项职责。也就是说,类 CreditCard 是否具有读卡操作?在此实例中,回答是肯定的。遍历对象-关系模型,确保所有此类连接都是有效的。

3. 反转连接,确保每个提供服务的协作者都从合理的地方收到请求。例如,如果 CreditCard 类收到了来自 CreditSale 类的请求 purchase amount,那么就有问题了。CreditCard 不知道购物金额是多少。

526

4. 使用步骤 3 中反转后的连接,确定是否真正需要其他类,或者职责在类之间的组织是否合适。

5. 确定是否可以将广泛请求的多个职责组合为一个职责。例如,读信用卡和取得授权在每一种情形下都会发生,可以将这两个职责组合为验证信用请求(validate credit request)职责,此职责包括取得信用卡号和取得授权。

可以将步骤 1~步骤 5 反复应用到每个类及需求模型的每一次评估中。

一旦创建了设计模型(第 12~18 章),就可以进行系统设计和对象设计的评审了。系统设计描述总体的产品体系结构、组成产品的子系统、将子系统分配给处理器的方式、将类分配给子系统的方式以及用户界面的设计。对象模型描述每个类的细节以及实现类之间的协作所必需的消息传送活动。

系统设计评审是这样进行的:检查面向对象分析期间所开发的对象-行为模型,并将所需的系统行为映射到为完成此行为而设计的子系统上。在系统行为的范畴内也要对并发和任务分配进行评审。对系统的行为状态进行评估以确定并发行为。使用用例进行用户界面设计。

对照对象-关系网检查对象模型,确保所有的设计对象都包括必要的属性和操作,以实现为每个 CRC 索引卡片所定义的协作。另外,要对操作细节的详细规格说明(即实现操作的算法)进行评审。

527

24.3 面向对象测试策略

如在第 22 章讲到的,经典的软件测试策略从“小范围”开始,并逐步过渡到“软件整体”。用软件测试的行话来说(第 23 章),就是先从单元测试开始,然后过渡到集成测试,并以确认测试和系统测试结束。在传统的应用中,单元测试关注最小的可编译程序单元——子程序(例如,构件、模块、子程序、程序)。一旦完成了一个单元的单独测试,就将其集成到程序结构中,并进行一系列的回归测试,以发现模块的接口错误及由于加入新模块所引发的副作用。最后,将系统作为一个整体进行测试,确保发现需求方面的错误。

24.3.1 面向对象环境中的单元测试

考虑面向对象软件时,单元的概念发生了变化。封装是类和对象定义的驱动力,也就是说,每个类和类的每个实例(对象)包装了属性(数据)和操纵这些数据的操作(也称为方法或服务)。最小的可测试单元是封装了的类,而不是单独的模块。由于一个类可以包括很多不同的操作,并且一个特定的操作又可以是很多不同类的一部分,因此,单元测试的含义发生了巨大的变化。

关键点 在面向对象软件中,最小的可测试“单元”是类,类测试是由封装在类中的操作和类的状态行为驱动的。

我们已经不可能再独立地测试单一的操作了（独立地测试单一的操作是单元测试的传统观点），而是要作为类的一部分进行操作。例如，考虑在一个类层次中，为超类定义了操作 $X()$ ，并且很多子类继承了此操作。每个子类都使用操作 $X()$ ，但是此操作是在为每个子类所定义的私有属性和操作的环境中应用的。由于使用操作 $X()$ 的环境具有微妙的差异，因此，有必要在每个子类的环境中测试操作 $X()$ 。这就意味着在真空中测试操作 $X()$ （传统的单元测试方法）在面向对象的环境中是无效的。

528

面向对象软件的类测试等同于传统软件的单元测试^①。面向对象软件的类测试与传统软件的单元测试是不同的，传统软件的单元测试倾向于关注模块的算法细节和流经模块接口的数据，而面向对象软件的类测试由封装在类中的操作和类的状态行为驱动。

24.3.2 面向对象环境中的集成测试

由于面向对象软件不具有层次控制结构，因此传统的自顶向下和自底向上的集成策略是没有意义的。另外，由于“组成类的构件之间的直接和非直接的交互”[Ber93]，因此每次将一个操作集成到类中通常是不可能的。

面向对象系统的集成测试有两种不同的策略 [Bin94a]。第一种集成策略是基于线程的测试，将响应系统的一个输入或一个事件所需要的一组类集成到一起。每个线程单独集成和测试，并应用回归测试确保不产生副作用。第二种集成策略是基于使用的测试，通过测试那些很少使用服务器类的类（称为独立类）开始系统的构建。测试完独立类之后，测试使用独立类的下一层类（称为依赖类）。按照这样的顺序逐层测试依赖类，直到整个系统构建完成。与传统集成不同，在可能的情况下，这种策略避免了作为替换操作的驱动模块和桩模块的使用（第 23 章）。

关键点 面向对象的集成测试是对响应一个给定事件所需要的一组类进行测试。

簇测试 [McG94] 是面向对象软件集成测试中的一个步骤。通过设计试图发现协作错误的测试用例，对一簇协作类（通过检查 CRC 和对象 - 关系模型来确定）进行测试。

24.3.3 面向对象环境中的确认测试

在确认级或系统级，类连接的细节消失了。如传统的确认方法一样，面向对象软件的确认关注用户可见的动作和用户可以辨别的来自系统的输出。为了辅助确认测试的导出，测试人员应该拟定出用例（第 9 章和第 10 章），用例是需求模型的一部分，提供了最有可能发现用户交互需求方面错误的场景。

传统的黑盒测试方法（第 23 章）可用于驱动确认测试。另外，测试人员可以选择从对象 - 行为模型导出测试用例，也可以从创建的事件流图（OOA 的一部分）导出测试用例。

引述 我将测试人员看成是项目的护卫。开发人员专注于创造成功，而测试人员则守护在他们左右，使其免遭失败。

James Bach

24.4 面向对象测试方法

529

面向对象体系结构导致封装了协作类的一系列分层子系统的产生。每个系统成分（子系统和类）完成的功能都有助于满足系统需求。有必要在不同的层次上测试面向对象系统，以发现错误。在类相互协作以及子系统

① 面向对象类的测试用例设计方法在 24.4 ~ 24.6 节讨论。

穿越体系结构层通信时可能出现这些错误。

面向对象软件的测试用例设计方法还在不断改进,然而,对于面向对象测试用例的设计,Berard已经提出了总体方法[Ber93]:

1. 每个测试用例都应该被唯一地标识,并明确地与被测试的类相关联。
2. 应该叙述测试的目的。
3. 应该为每一个测试开发测试步骤,并包括以下内容:将要测试的类的指定状态列表;作为测试结果要进行检查的消息和操作列表;对类进行测试时可能发生的异常列表;外部条件列表(即软件外部环境的变更,为了正确地进行测试,这种环境必须存在);有助于理解或实现测试的补充信息。

面向对象测试与传统的测试用例设计是不同的,传统的测试用例是通过软件的输入-处理-输出视图或单个模块的算法细节来设计的,而面向对象测试侧重于设计适当的操作序列以检查类的状态。

24.4.1 面向对象概念的测试用例设计含义

经过分析模型和设计模型的演变,类成为了测试用例设计的目标。由于操作和属性是封装的,因此从类的外面测试操作通常是徒劳的。尽管封装是面向对象的重要设计概念,但它可能成为测试的一个小障碍。如Binder[Bin94a]所述:“测试需要报告对象的具体状态和抽象状态。”然而,封装使获取这些信息有些困难,除非提供内置操作来报告类的属性值,否则,可能很难获得一个对象的状态快照。

网络资源 一些极好的有关面向对象测试的论文集和资源可在<https://www.thecsiac.com>找到。

继承也为测试用例设计提出了额外的挑战。我们已经注意到,即使已取得复用,每个新的使用环境也需要重新测试。另外,由于增加了所需测试环境的数量,因此多重继承^①使测试进一步复杂化[Bin94a]。若将从超类派生的子类实例用于相同的问题域,则测试子类时,使用超类中生成的测试用例集是可能的。然而,若子类用在一个完全不同的环境中,则超类的测试用例将具有很小的可应用性,因而必须设计新的测试用例集。

530

24.4.2 传统测试用例设计方法的可应用性

第23章描述的白盒测试方法可以应用于类中定义的操作。基本路径、循环测试或数据流技术有助于确保一个操作中的每条语句都测试到。然而,许多类操作的简洁结构使某些人认为:用于白盒测试的工作投入最好直接用于类层次的测试。

与利用传统的软件工程方法所开发的系统一样,黑盒测试方法也适用于面向对象系统。如我们在第23章提到的,用例可为黑盒测试和基于状态的测试设计提供有用的输入。

24.4.3 基于故障的测试^②

在面向对象系统中,基于故障的测试目标是设计测试以使其最有可能发现似乎可能出现的故障(以下称为似然故障)。由于产品或系统必须符合

关键点 基于故障的测试策略是假设一组似乎可能出现的故障,然后导出测试去证明每个假设。

① 应非常小心使用的一个面向对象概念。

② 24.4.3节和24.4.4节是从Brain Marick发布在因特网新闻组comp.testing上的文章中摘录的,已得到作者的许可。有关该主题的详细信息见[Mar94]。应该注意到,24.4.3节和24.4.4节讨论的技术也适用于传统软件。

客户需求，因此完成基于故障的测试所需的初步计划是从分析模型开始的。测试人员查找似然故障（即系统的实现中有可能产生错误的方面）。为了确定这些故障是否存在，需要设计测试用例以检查设计或代码。

当然，这些技术的有效性依赖于测试人员如何理解似然故障。若在面向对象系统中真正的故障被理解为“没有道理”的，则这种方法实际上并不比任何随机测试技术好。然而，若分析模型和设计模型可以洞察有可能出错的事物，则基于故障的测试可以花费相当少的工作量而发现大量的错误。

集成测试寻找的是操作调用或信息连接中的似然错误。在这种环境下，可以发现三种错误：非预期的结果，使用了错误的操作/消息，以及不正确的调用。为确定函数（操作）调用时的似然故障，必须检查操作的行为。

提问 在操作调用和消息连接中会遇到哪些类型的故障？

集成测试适用于属性，同样也适用于操作。对象的“行为”通过赋予属性值来定义。测试应该检查属性以确定不同类型的对象行为是否存在合适的值。

集成测试试图发现用户对象而不是服务对象中的错误，注意到这一点很重要。用传统的术语来说，集成测试的重点是确定调用代码而不是被调用代码中是否存在错误。以操作调用为线索，这是找出调用代码的测试需求的一种方式。

531

24.4.4 基于场景的测试设计

基于故障的测试忽略了两种主要类型的错误：（1）不正确的规格说明；（2）子系统间的交互。当出现了与不正确的规格说明相关的错误时，产品并不做客户希望的事情，而是有可能做错误的事情或漏掉重要的功能。但是，在这两种情况下，质量（对需求的符合性）均会受到损害。当一个子系统的行为创建的环境（例如事件、数据流）使另一个子系统失效时，则出现了与子系统交互相关的错误。

基于场景的测试关心用户做什么，而不是产品做什么。这意味着捕获用户必须完成的任务（通过用例），然后在测试时使用它们及其变体。

场景可以发现交互错误。为了达到这个目标，测试用例必须比基于故障的测试更复杂且更切合实际。基于场景的测试倾向于用单一测试检查多个子系统（用户并不限制自己一次只用一个子系统）。

关键点 基于场景的测试将发现任何角色与软件进行交互时出现的错误。

24.5 类级可应用的测试方法

“小范围”测试侧重于单个类及该类封装的方法。面向对象测试期间，随机测试和分割是用于检查类的测试方法。

24.5.1 面向对象类的随机测试

为简要说明这些方法，考虑一个银行应用，其中 Account 类有下列操作：open()、setup()、deposit()、withdraw()、balance()、summarize()、creditLimit() 及 close()[Kir94]。其中，每个操作均可应用于 Account 类，但问题的本质隐含了一些限制（例如，账号必须在其他操作可应用之前打开，在所有操作完成之后关闭）。即使有了这些限制，仍存在很多种操作排列。一个 Account 对象的最小行为的生命历史包含以下操作：

建议 随机测试可能的排列数可以变得相当大。与正交数组测试相类似的策略可以用于提高测试的有效性。


```
open•setup•deposit•withdraw•close
```

这表示 Account 的最小测试序列。然而，可以在这个序列中发生大量其他行为：

```
open•setup•deposit•[deposit|withdraw|balance|summarize|creditLimit]n•withdraw•close
```

532

可以随机产生一些不同的操作序列，例如：

测试用例 r_1 : open•setup•deposit•deposit•balance•summarize•withdraw•close

测试用例 r_2 : open•setup•deposit•withdraw•deposit•balance•creditLimit•withdraw•close

执行这些序列和其他随机顺序测试，以检查不同类实例的生命历史。

SafeHome 类测试

[场景] Shakira 的工作间。

[人物] Jamie 与 Shakira, SafeHome 软件工程团队成员，负责安全功能的测试用例设计。

[对话]

Shakira：我已经为 Detector 类（图 14-4）开发了一些测试，你知道，这个类允许访问安全功能的所有 Sensor 对象。你熟悉它吗？

Jamie（笑）：当然，它允许你加入“小狗焦虑症”传感器。

Shakira：而且是唯一的一个。不管怎么样，它包含 4 个操作的接口：read()、enable()、disable() 和 test()。在传感器可读之前，它必须被激活。一旦激活，就可以进行读和测试，且随时可以中止它，除非正在处理警报条件。因此，我定义了检查其行为生命历史的简单测试序列（向 Jamie 展示下述序列）。

```
#1: enable•test•read•disable
```

Jamie：不错。但你还得做更多的测试！

Shakira：我知道。这里有我提出的其他测

试序列。

（向 Shakira 展示下述序列）

```
#2: enable•test*[read]n•test•disable
```

```
#3: [read]n
```

```
#4: enable•disable•[test | read]
```

Jamie：看我能否理解这些测试序列的意图。#1 通过一个正常的生命历史，属于常规使用。#2 重复 read() 操作 n 次，那是一个可能出现的场景。#3 在传感器激活之前尽力读取它……那应该产生某种错误信息，对吗？#4 激活和中止传感器，然后尽力读取它，这与 #2 不是一样的吗？

Shakira：实际上不一样。在 #4 中，传感器已经被激活，#4 测试的实际上是 disable() 操作是否像其预期的一样有效工作。disable() 之后，read() 或 test() 应该产生错误信息。若没有，则说明 disable() 操作中有错误。

Jamie：太棒了，记住这 4 个测试必须应用于每一类传感器，因为所有这些操作根据其传感器类型可能略有不同。

Shakira：不用担心，那是计划之中的事。

24.5.2 类级的划分测试

与传统软件的等价划分（第 23 章）基本相似，划分测试（partition testing）可减小测试特定类所需的测试用例数量。对输入和输出进行分类，设计测试用例以检查每个分类。但划分类别是如何得到的呢？

基于状态的划分就是根据它们改变类状态的能力对类操作进行分类。例如，考虑 Account 类，状态操作包括 deposit() 和 withdraw()，而非状态操作包括

提问 在类级上什么测试选项可供使用？

533

balance()、summarize() 和 creditLimit()。将改变状态的操作和不改变状态的操作分开，分别进行测试，因此：

测试用例 p_1 : open•setup•deposit•deposit•withdraw•withdraw•close
测试用例 p_2 : open•setup•deposit•summarize•creditLimit•withdraw•close

测试用例 p_1 检查改变状态的操作，而测试用例 p_2 检查不改变状态的操作（除了那些最小测试序列中的操作）。

也可以应用其他类型的划分测试。基于属性的划分就是根据它们所使用的属性对类操作进行分类。基于类别的划分就是根据每个操作所完成的一般功能对类操作进行分类。

24.6 类间测试用例设计

当开始集成面向对象系统时，测试用例的设计变得更为复杂。在这个阶段必须开始类间协作的测试。为说明“类间测试用例生成”[Kir94]，我们扩展 24.5 节中讨论的银行例子，让它包括图 24-2 中的类与协作。图中箭头的方向指明消息传递的方向，标注则指明作为消息隐含的协作的结果而调用的操作。

引述 对于面向对象的开发，定义单元测试和集成测试范围的边界是不同的。在这个过程中，可以在很多点上设计和检查测试。因此，“设计一点儿，编码一点儿”变成了“设计一点儿，编码一点儿，测试一点儿”。
Robert Binder

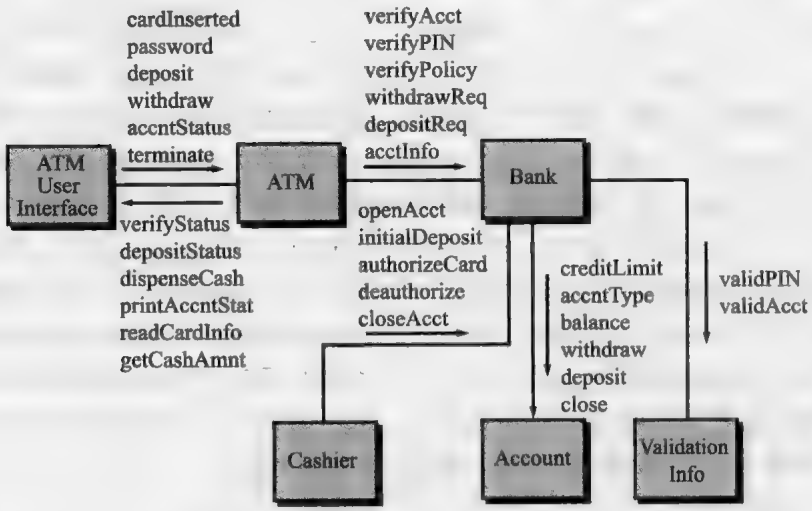


图 24-2 银行应用的类协作图 [Kir94]

与单个类的测试相类似，类协作测试可以通过运用随机和划分方法、基于场景测试及行为测试来完成。

24.6.1 多类测试

Kirani 和 Tsai[Kir94] 提出了利用下列步骤生成多类随机测试用例的方法：

1. 对每个客户类，使用类操作列表来生成一系列随机测试序列。这些操作将向其他服务类发送消息。
2. 对生成的每个消息，确定协作类和服务对象中的相应操作。
3. 对服务对象中的每个操作（已被来自客户对象的消息调用），确定它传送的消息。
4. 对每个消息，确定下一层被调用的操作，并将其引入到测试序列中。

为便于说明 [Kir94]，考虑 Bank 类相对于 ATM 类的操作序列（图 24-2）：

```
verifyAcct•verifyPIN•[[verifyPolicy•withdrawReq]|depositReq|acctInfoREQ]n
```

Bank 类的一个随机测试用例可以是：

```
测试用例  $r_3$ : verifyAcct•verifyPIN•depositReq
```

为考虑涉及该测试的协作者，考虑与测试用例 r_3 中提到的操作相关的消息。为了执行 verifyAcct() 与 verifyPIN()，Bank 类必须与 ValidationInfo 类协作。为了执行 depositReq()，Bank 类必须与 Account 类协作。因此，检查这些协作的新测试用例为：

测试用例 r_4 ：

```
Test case  $r_4$  = verifyAcct [Bank:validAcctValidationInfo]•verifyPIN  
[Bank: validPinValidationInfo]•depositReq [Bank: depositaccount]
```

多个类的划分测试方法与单个类的划分测试方法类似，单个类的划分测试方法如在 24.5.2 节讨论的那样。然而，可以对测试序列进行扩展，以包括那些通过发送给协作类的消息而激活的操作。另一种划分测试方法基于特殊类的接口。参看图 24-2，Bank 类从 ATM 类和 Cashier 类接收消息，因此，可以通过将 Bank 类中的操作划分为服务于 ATM 类的操作和服务于 Cashier 类的操作对其进行测试。基于状态的划分（24.5.2 节）可用于进一步细化上述划分。

535

24.6.2 从行为模型导出的测试

在第 11 章中，我们已讨论过用状态图表示类的动态行为模型。类的状态图可用于辅助生成检查类（以及与该类的协作类）的动态行为的测试序列。图 24-3[Kir94] 给出了前面讨论的 Account 类的状态图。根据该图，初始变换经过了 Empty acct 状态和 Setup acct 状态，该类实例的绝大多数行为发生在 Working acct 状态。最终的 Withdrawal 和结束账户操作使得 Account 类分别向 Nonworking acct 状态和 Dead acct 状态发生转换。

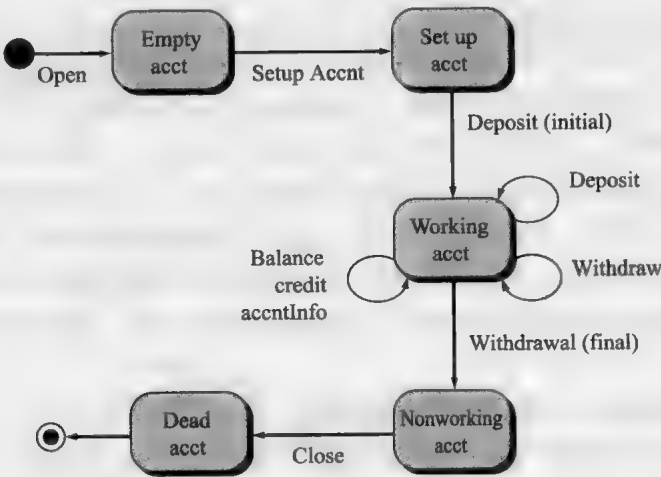


图 24-3 Account 类的状态转换图 [Kir94]

将要设计的测试应该覆盖所有的状态，也就是说，操作序列应该使 Account 类能够向所有可允许的状态转换：

```
测试用例  $s_1$ : open•setupAcct•deposit (initial)•withdraw (final)•close
```

应该注意到，这个序列与 24.5.2 节所讨论的最小测试序列相同。下面将其他测试序列加入最小测试序列中：

测试用例 s_2 : `open•setupAccnt•deposit(initial)•deposit•balance•credit•withdraw
(final)•close`

536

测试用例 s_3 : `open•setupAccnt•deposit(initial)•deposit•withdraw•acctInfo•withdr
aw (final)•close`

可以设计更多的测试用例以保证该类的所有行为已被充分检查。在该类的行为与一个或多个类产生协作的情况下，可以用多个状态图来追踪系统的行为流。

可以通过“广度优先”[McG94]的方式来遍历状态模型。在这里，广度优先意味着一个测试用例检查单个转换，之后在测试新的转换时，仅使用前面已经测试过的转换。

考虑银行系统中的一个 CreditCard 对象。CreditCard 对象的初始状态为 undefined（即未提供信用卡号）。在销售过程中一旦读取信用卡，对象就进入了 defined 状态，即属性 card number、expiration date 以及银行专用的标识符被定义。当信用卡被发送以请求授权时，它处于 submitted 状态，当接收到授权时，它处于 approved 状态。可以通过设计使转换发生的测试用例来测试 CreditCard 对象从一个状态到另一个状态的转换。对这种测试类型的广度优先方法在检查 undefined 和 defined 之前不会检查 submitted 状态。若这样做了，它就使用了尚未经过测试的转换，从而违反了广度优先准则。

24.7 小结

面向对象测试的总体目标是以最少的工作量发现最多的错误，这与传统软件的测试目标是一样的。但是，面向对象测试的策略和战术与传统软件有很大差异。测试的视角扩展到了需求模型和设计模型，另外，测试的重点从过程构件（模块）移向了类。

由于面向对象需求模型、设计模型和最终的源代码在语义上是有联系的，在建模活动期间，测试（以技术评审的形式）就开始了。因此，可以将 CRC、对象－关系模型、对象－行为模型的评审看成是第一阶段的测试。

一旦有了代码，就对每个类进行单元测试。可以使用多种方法对类测试进行设计：基于故障的测试、随机测试和划分测试。每种方法都检查类中封装的操作。设计测试序列以保证对相关的操作进行了检查。通过检查类的状态（由属性值表示）以确定是否存在错误。

集成测试可以通过基于线程或基于使用的策略来完成。基于线程的测试将为了响应一个输入或事件而相互协作的一组类集成在一起。基于使用的测试从那些不使用服务类的类开始，以分层的方式构建系统。集成测试用例设计方法也使用随机测试和划分测试。另外，基于场景和从行为模型中生成的测试可用于测试类及其协作关系。测试序列追踪类间协作的操作流。

537

面向对象系统的确认测试是面向黑盒的测试，可以应用传统软件中所讨论的黑盒方法来完成。然而，基于场景的测试在面向对象系统的确认中占优势，使用例成为确认测试的主要驱动者。

习题与思考题

24.1 用自己的话描述为什么在面向对象系统中类是最小的合理测试单元。

24.2 若现有类已进行了彻底的测试，为什么我们还是必须对从现有类实例化的子类进行重新测试？

我们可以使用为现有类设计的测试用例吗?

- 24.3 为什么“测试”应该从面向对象分析和设计开始?
- 24.4 为 SafeHome 导出一组 CRC 索引卡片,按照 24.2.2 节讲述的步骤确定是否存在不一致性。
- 24.5 基于线程和基于使用的集成测试策略有什么不同?簇测试如何适应?
- 24.6 将随机测试和划分方法运用到设计 SafeHome 系统时定义的两个类。产生用于展示操作调用序列的测试用例。
- 24.7 运用多类测试及从 SafeHome 设计的行为模型中生成的测试。
- 24.8 运用随机测试、划分方法、多类测试及 24.5 节和 24.6 节所描述的银行应用的行为模型导出的测试,再另外生成 4 个测试。

扩展阅读与信息资源

第 22 章和第 23 章的推荐读物与阅读信息部分所列出的很多测试方面的书都在一定程度上讨论了面向对象系统的测试。Bashir 和 Goel (《 Testing Object-Oriented Software 》, Springer, 2012)、Schach (《 Object-Oriented and Classical Software Engineering 》, McGraw-Hill, 8th ed., 2010) 以及 Bruege 和 Dutoit (《 Object-Oriented Software Engineering Using UML, Patterns, and Java 》, Prentice Hall, 3rd ed., 2009) 在更广的软件工程实践环境中考虑面向对象测试。Jorgensen (《 Software Testing: A Craftsman ’ s Approach 》, Auerbach, 3rd ed., 2008) 讨论了形式化技术和面向对象技术。Yurga (《 Testing and Testability of Object-Oriented Software Systems via Metrics: A Metrics-Based Approach to the Testing Process and Testability of Object-Oriented Software Systems 》, LAP Lambert, 2011)、Sykes 和 McGregor (《 Practical Guide to Testing Object-Oriented Software 》, Addison-Wesley, 2001)、Binder (《 Testing Object-Oriented Systems 》, Addison-Wesley, 1999) 以及 Kung 和他的同事 (《 Testing Object-Oriented Software 》, Wiley-IEEE Computer Society Press, 1998) 都非常详细地描述了面向对象测试。Freeman 和 Pryce (《 Growing Object-Oriented Software, Guided by Tests 》, Addison-Wesley, 2009) 讨论了面向对象软件的测试驱动设计。Denney (《 Use Case Levels of Test: A Four-Step Strategy for Building Time and Innovation in Software Test Design 》, CreateSpace Independent Publishing, 2012) 讨论了可以应用于测试面向对象系统的技术。

从网上可以获得大量有关面向对象测试方法的信息。最新的有关测试技术的参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

测试 WebApp

要点浏览

概念: WebApp 测试是一组相关的活动，这些活动都具有共同的目标：发现 WebApp 在内容、功能、可用性、导航性、性能、容量及安全方面存在的错误。为实现这个目标，要同时应用包括评审及运行测试的测试策略。

人员: 所有参加 WebApp 测试的 Web 工程师和项目的其他利益相关者（经理、客户、最终用户）。

重要性: 如果最终用户遇到错误，就会动摇对 WebApp 的信心，他们会转向其他地方寻找需要的内容及功能，WebApp 就会失败。因此，Web 工程师一定要在 WebApp 上线前尽可能多地排除错误。

步骤: 在进行 WebApp 测试时，首先关注

用户可见的方面，之后进行技术及内部结构方面的测试。要进行 7 个步骤的测试：内容测试、界面测试、导航测试、构件测试、配置测试、性能测试及安全测试。

工作产品: 在某些情况下，需要制定 WebApp 测试计划。在每种情况下，都需要为每一个测试步骤开发一组测试用例，并且要对记录测试结果的文档进行维护，以备将来使用。

质量保证措施: 虽然你从来都不能保证你已经完成了需要的每一项测试，但是，你能够肯定测试已经发现了错误（并且已经改正）。此外，如果你已经制定了一份测试计划，就可以对照测试计划进行检查，以确保所有计划的测试都已经完成。

WebApp 项目的工作普遍都很紧迫。利益相关者由于担心来自其他 WebApp 的竞争，迫于客户的要求，并担心他们将失去市场，因而迫使 WebApp 仓促上线。其结果是，在 Web 开发过程中，技术活动通常开始较晚，例如，有时给 WebApp 测试所剩的时间很短，这可能是一个灾难性的错误。为了避免这种错误的发生，Web 工程团队一定要确保每个工作产品都具有高质量。Wallace 和他的同事 [Wal03] 在谈及这一点时讲道：

不应该等到项目完成时才进行测试。在你写第一行代码之前就开始测试。进行持续及有效的测试，这样你将开发出更耐用的 Web 站点。

由于不能在传统意义上对需求模型及设计模型进行测试，因此除了可运行的测试，Web 工程团队还应该进行正式技术评审（第 20 章），目的是在 WebApp 交付最终用户使用之前发现并改正错误。

关键概念

兼容性测试
构件级测试
配置测试
内容测试
数据库测试
负载测试
导航测试
性能测试
计划
质量维度
安全性测试
策略
压力测试
可用性测试
用户界面测试

25.1 WebApp 的测试概念

测试是为了发现（并最终改正）错误而运行软件的过程。这个首次在

第 22 章介绍的基本原理对 WebApp 也是一样的。事实上, 由于基于 Web 的系统及应用位于网络上, 并与很多不同的操作系统、浏览器 (或其他个人通信设备)、硬件平台、通信协议及“暗中的”应用进行交互作用, 因此错误的查找是一个重大的挑战。

为了了解 Web 工程环境中的测试目标, 我们必须考虑 WebApp 质量的多种维度^①。在此讨论中, 我们考虑在讨论 Web 开发的测试工作中都特别关注的质量维度, 同时, 我们也考虑作为测试结果所碰到的错误的特性以及为发现这些错误所采用的测试策略。

25.1.1 质量维度

良好的设计应该将质量集成到 WebApp 中。通过对设计模型中的不同元素进行一系列技术评审, 并应用本章所讨论的测试过程, 对质量进行评估。评估和测试都要检查下面质量维度中的一项或多项 [Mil00a]:

- **内容。**在语法及语义层对内容进行评估。在语法层, 对基于文本的文档进行拼写、标点及文法方面的评估; 在语义层, (所表示信息的) 正确性、(整个内容对象及相关对象的) 一致性及清晰性都要评估。
- **功能。**对功能进行测试, 以发现与客户需求不一致的错误。对每一项 WebApp 功能, 都要评定其正确性、不稳定性及与相应的实现标准 (例如, Java 或 AJAX 语言标准) 的总体符合程度。
- **结构。**对结构进行评估, 以保证它能正确地表示 WebApp 的内容及功能, 具备可扩展性, 并支持新内容、新功能的增加。
- **可用性。**对可用性进行测试, 以保证接口支持各种类型的用户, 各种用户都能够学会及使用所有需要的导航语法及语义。
- **导航性。**对导航性进行测试, 以保证检查所有的导航语法及语义, 发现任何导航错误 (例如, 死链接、不合适的链接、错误链接)。
- **性能。**在各种不同的操作条件、配置及负载下, 对性能进行测试, 以保证系统响应用户的交互并处理极端的负载情况, 而且没有出现操作上不可接受的性能下降。
- **兼容性。**在客户端及服务器端, 在各种不同的主机配置下通过运行 WebApp 对兼容性进行测试, 目的是发现针对特定主机配置的错误。
- **互操作性。**对互操作性进行测试, 以保证 WebApp 与其他应用和数据库有正确接口。
- **安全性。**对安全性进行测试, 通过评定可能存在的弱点, 试图对每个弱点进行攻击。任何一次成功的突破都被认为是一个安全漏洞。

人们已经制定了 WebApp 测试的策略及多种战术, 用来测试这些质量维度, 在本章后面将对这些进行讨论。

25.1.2 WebApp 环境中的错误

对于成功的 WebApp 测试, 它们所遇到的错误具有很多独特的特点 [Ngu00]:

1. 由于 WebApp 测试发现的很多类型的错误都首先表现在客户端中

提问 我们如何在 WebApp 及其所处的环境中评定质量?

引述 创新对于软件测试人员是苦乐参半的事情。当我们似乎知道如何测试一项特定的技术时, 恰好出现了一项新技术 (WebApp), 以前的好办法都不灵了。

James Bach

541

提问 是什么使得在 WebApp 运行中遇到的错误不同于传统软件中遇到的错误?

^① 第 19 章讨论的通用软件质量维度同样可以应用于 WebApp。

(即通过在特定浏览器或个人通信设备上实现的接口),因此 Web 工程师看到了错误的征兆,而不是错误本身。

2. 由于 WebApp 是在很多不同的配置及不同的环境中实现的,因此要在最初遇到错误的环境之外再现错误可能是很困难的,或是不可能的。
3. 虽然许多错误是不正确的设计或不合适的 HTML(或其他程序设计语言)编码的结果,但很多错误的原因都能够追溯到 WebApp 配置。
4. 由于 WebApp 位于客户/服务器体系结构之中,因此在三层体系结构(客户、服务器或网络本身)中追踪错误是很困难的。
5. 某些错误应归于静态的操作环境(即进行测试的特定配置),而另外一些错误可归于动态的操作环境(即瞬间的资源负载或时间相关的错误)。

542

上述 5 个错误特点说明:在 WebApp 测试的错误诊断过程中,环境起着非常重要的作用。在某些情况(例如,内容测试)下,错误的位置是明显的;但对于很多其他类型的 WebApp 测试(例如,导航测试、性能测试、安全测试),错误的根本原因很难确定。

25.1.3 测试策略

WebApp 测试策略采用所有软件测试所使用的基本原理(第 22 章),并建议使用面向对象系统所使用的策略和战术(第 24 章)。下面的步骤对此方法进行了总结:

1. 对 WebApp 的内容模型进行评审,以发现错误。
2. 对接口模型进行评审,保证其适合所有的用例。
3. 评审 WebApp 的设计模型,发现导航错误。
4. 测试用户界面,发现表现机制和导航机制中的错误。
5. 对所选择的功能构件进行单元测试。
6. 对贯穿体系结构的导航进行测试。
7. 在各种不同的环境配置下实现 WebApp,并测试 WebApp 对于每一种配置的兼容性。
8. 进行安全性测试,试图攻击 WebApp 或其所处环境的弱点。
9. 进行性能测试。
10. 通过可监控的最终用户群对 WebApp 进行测试;对他们与系统的

交互结果进行以下方面的评估,包括内容和导航错误、可用性、兼容性、WebApp 的安全性、可靠性及性能等方面的评估。

由于很多 WebApp 在不断进化,因此 WebApp 测试是 Web 支持人员所从事的一项持续活动,他们使用回归测试,这些测试是从首次开发 WebApp 时所开发的测试中导出的。

25.1.4 测试计划

对某些 Web 开发人员来说,使用计划一词(在任何环境中)是不受欢迎的。这些开发人员不做计划,而只是抱着这样的想法开始工作——希望开发一个招人喜爱的 WebApp。更严格的方法则认识到计划工作建立了所有工作遵循的路线图,这种努力是值得的。Splain 和 Jaskiel[Spl01]在他们关于 WebApp 测试的书中谈道:

543

除了最简单的网站以外,其他 WebApp 对某种测试计划的需要很快就变得很明显。通常,在随机测试中发现的最初的错误数量很大,以致于在第一次检测到错误时不能对所有的

关键点 WebApp 测试的总体策略在这里可以总结为 10 个步骤。

网络资源 WebApp 测试方面的优秀文章可以在 www.stickyminds.com/testing.asp 找到。

错误进行定位，这就增加了 Web 站点或 WebApp 测试人员的负担。他们不仅要幻想虚构的新测试，还必须记住以前的测试是如何运行的，以对 Web 站点或应用进行可靠的重复测试，并确保已知的错误都被排除，同时没有引入新的错误。

每位 Web 工程师面临的问题是：我们如何“幻想虚构的新测试”，这些测试应该集中在什么地方？对这些问题的回答都包含在测试计划中。

WebApp 的测试计划确定了：（1）当测试开始时所应用的任务集合[⊖]；（2）执行每项测试任务所生产的工作产品；（3）以何种方式对测试结果进行评估、记录，以及在进行回归测试时如何重用。在某些情况下，测试计划被集成到项目计划中；而有些情况下测试计划是单独的文档。

25.2 测试过程概述

在对 Web 工程进行测试时，首先测试最终用户能够看到的内容和界面。随着测试的进行，再对体系结构及导航设计的各个方面进行测试。最后，测试的焦点转到测试技术能力——WebApp 基础设施及安装或实现方面的问题，这些方面对于最终用户并不总是可见的。

图 25-1 将 WebApp 的测试过程与 WebApp 的设计金字塔（第 17 章）相并列。需要注意的是，当测试流从左到右、从上到下移动时，首先测试 WebApp 设计中的用户可见元素（金字塔的顶端元素），之后对内部结构的设计元素进行测试。

关键点 测试计划确定了测试任务集和将被开发的工作产品，以及结果被评估、记录及重用的方式。

引述 一般情况下，在其他应用中使用的软件测试技术与在基于 Web 的应用中使用的软件测试技术相同……两种测试类型的不同之处在于 Web 环境中的技术变量增加了。
Hung Nguyen

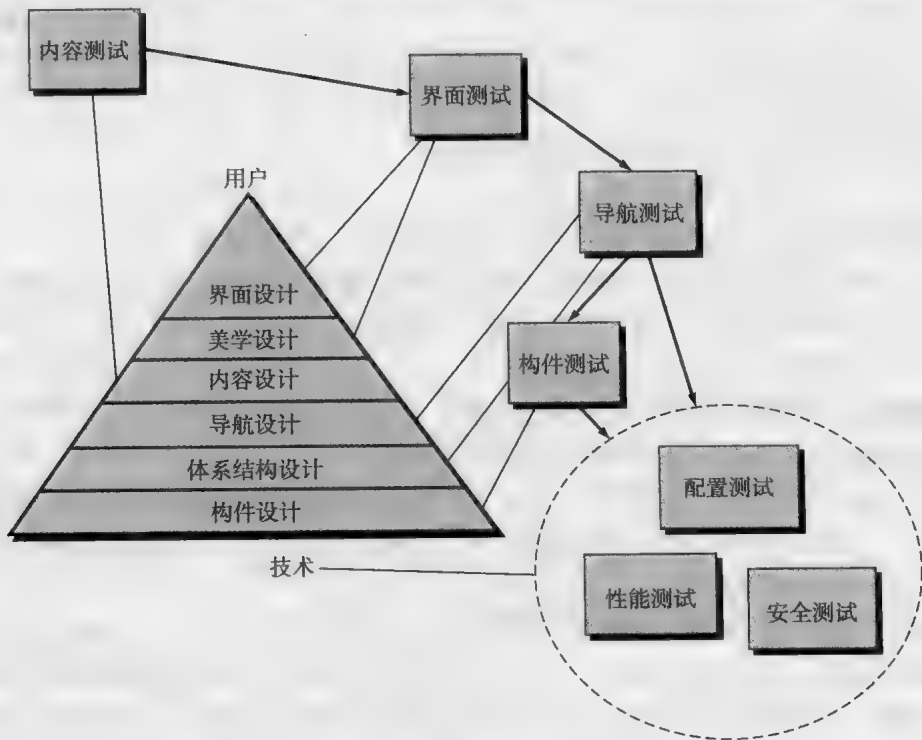


图 25-1 测试过程

⊖ 第 3 章对任务集合进行了讨论。在本书中也使用相关的术语——“工作流”——来描述完成软件工程活动所需要的一系列任务。

25.3 内容测试

WebApp 内容中的错误可以小到打字错误，也可以大到不正确的信息、不合适的组织、或者违背知识产权法。内容测试试图在用户碰到这些问题及很多其他问题之前就发现它们。

内容测试结合了评审和可运行的测试用例的生成。采用评审来发现内容中的语义错误（在 25.3.1 节讨论）；可运行的测试用于发现内容错误，这些错误可被跟踪到动态导出的内容（这些内容由从一个或多个数据库中获取的数据驱动）。

25.3.1 内容测试的目标

内容测试具有三个重要的目标：（1）发现基于文本的文档、图形表示和其他媒体中的语法错误（例如，打字错误、文法错误）；（2）发现当导航发生时所展现的任何内容对象中的语义错误（即信息的准确性和完备性方面的错误）；（3）发现展示给最终用户的内容的组织或结构方面的错误。

为了达到第一个目标，可以使用自动的拼写和语法检查。然而，很多语法上的错误会逃避这种工具的检查，而必须由审查人员（测试人员）人为发现。实际上，大型网站会借助专业稿件编辑器，以发现打字错误、语法错误、内容一致性错误、图形表示错误和交叉引用错误。

语义测试关注每个内容对象所显示的信息。评审人员（测试人员）必须回答以下问题：

- 信息确实准确吗？
- 信息简洁扼要吗？
- 内容对象的布局对于用户来说容易理解吗？
- 嵌入在内容对象中的信息易于发现吗？
- 对于从其他地方导出的所有信息，是否提供了合适的引用？
- 显示的信息是否是内部一致的？与其他内容对象中所显示的信息是否一致？
- 内容是否具有攻击性？是否容易误解？或者是否会引起诉讼？
- 内容是否侵犯了现有的版权或商标？
- 内容是否包括补充现有内容的内部链接？链接正确吗？
- 内容的美学风格是否与界面的美学风格相矛盾？

对于大型的 WebApp（包含成百上千个内容对象）来说，要获得所有这些问题的答案可能是一项令人生畏的任务。然而，不能发现语义错误将动摇用户对 WebApp 的信任，并且会导致基于 Web 的 App 的失败。

内容对象存在于具有特定风格的体系结构之中（第 17 章）。在内容测试期间，要对内容体系结构的结构及组织进行测试，以确保将所需要的内容以合适的顺序和关系展现给最终用户。例如，SafeHomeAssured.com WebApp 显示了关于传感器的多种信息，其中传感器是安全和监视产品的一部分。内容对象提供描述信息、技术规格说明、照片和相关的信息。SafeHomeAssured.com 内容体系结构的测试试图发现这种信息的表示方面的错误（例如，用传感器 Y 的照片来描述传感器 X）。

建议 虽然正式的技术评审不是测试的组成部分，但应执行内容评审，以确保内容的质量。

关键点 内容测试的目标包括：
（1）发现内容中的语法错误；
（2）发现语义错误；
（3）发现结构错误。

提问 要发现内容中的语义错误，应该提出和回答哪些问题？

545

546

软件工具 Web 内容测试工具

[目标] Web 内容测试工具的目标是识别错误，这些错误会阻止 Web 页以可读和有组织的方式显示 Web 内容。

[机制] 这些工具通常提示输入 Web 资源的 URL 进行测试。每个工具提供错误列表（例如，没有遵循标记语言标准）及如何改正这些错误的建议。

[代表性工具]^①

- <http://validator.w3.org/>，在线的 WC3 工

具，可检查 Web 页标记语言的有效性（HTM，XHTML，SMIL，MathML）。

- <http://jigsaw.w3.org/css-validator/>，在线的 WC3 工具，可检查 CSS 样式表，并用 CSS 样式表检查文档。
- <http://validator.w3.org/feed/>，在线的 WC3 工具，可检查 Atom 语法或 RSS 文件。

25.3.2 数据库测试

现代 WebApp 要比静态的内容对象做更多的事情。在很多应用领域中，WebApp 要与复杂的数据库管理系统接口，并构建动态的内容对象，这种对象是使用从数据库中获取的数据实时创建的。

例如，用于金融服务的 WebApp 能够产生某种特殊产权（例如，股票或互惠基金）的复杂的文本信息、表格信息和图形信息。当用户已经申请了某种特殊的产权信息之后，就会自动创建表示这种信息的复合内容对象。为了完成此任务，需要下面的步骤：（1）查询股票数据库；（2）从数据库中抽取相关的数据；（3）抽取的数据一定要组织为一个内容对象；（4）将这个内容对象（代表由某个最终用户请求的定制信息）传送到客户环境显示。每个步骤的结果都可能发生错误，并且一定会发生。数据库测试的目标就是发现这些错误，然而，数据库测试会由于以下多种原因而变得复杂：

1. 客户端请求的原始信息很少以能够输入数据库管理系统（DBMS）的形式（例如，结构化查询语言 SQL）表示出来。因此，应该设计测试，用来发现将用户的请求翻译成 DBMS 可处理格式的过程中所产生的错误。
2. 数据库可能离装载 WebApp 的服务器很远。因此，应该设计测试，用来发现 WebApp 和远程数据库之间的通信所存在的错误^②。
3. 从数据库中获取的原始数据一定要传递给 WebApp 服务器，并且被正确地格式化，以便随后传递给客户端。因此，应该设计测试，用来证明 WebApp 服务器接收到的原始数据的有效性，并且还要生成另外的测试，证明转换的有效性，将这种转换应用于原始数据，能够生成有效的内容对象。
4. 动态内容对象一定以能够显示给最终用户的形式传递给客户端。因

提问 哪些问题使 WebApp 的数据库测试变得复杂了？

引述 对于频繁停机、在事务处理的中途停止或可用性差的网站，我们不可能有信心。因此，测试在整个开发过程中具有关键作用。

Wing Lam

547

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

② 当遇到分布式数据库，或者需要访问数据仓库（第1章）时，这些测试可能变得非常复杂。

此,应该设计一系列测试,用来发现内容对象格式方面的错误,以及测试与不同的客户环境配置的兼容性。

考虑这4种因素,对图25-2中所列出的每一个“交互层”[Ngu01],都应该使用测试用例的设计方法。测试应该保证:(1)有效信息通过界面层在客户与服务器之间传递;(2)WebApp 正确地处理脚本,并且正确地抽取或格式化用户数据;(3)用户数据被正确地传递给服务器端的数据转换功能,此功能格式化为合适的查询(例如,SQL);(4)查询被传递到数据管理层^①,此层与数据库访问程序(很可能位于另一台机器)通信。

通常使用可复用的构件来构造图25-2所示的数据转换层、数据管理层和数据库访问层,这些可复用的构件都分别进行了合格性确认,并且被打成一个包。如果是这种情况,那么WebApp的测试便集中在图25-2所示的客户层与前两个服务器层(WebApp和数据转换)之间交互的测试用例的设计。

应该对用户界面层进行测试,以确保对每一个用户查询都正确地构造了HTML脚本,并且正确地传输给服务器端。还应该对服务器端的WebApp层进行测试,以确保能够从HTML脚本中正确地抽取用户数据,并且正确地传输给服务器端的数据转换层。

应该对数据转换功能进行测试,以确保创建了正确的SQL,并且传给合适的数据管理构件。

为合理地设计这些数据库测试所需要了解的主要技术的详细讨论超出了本书的范围。感兴趣的读者应参看[Sce02]、[Ngu01]和[Bro01]。

25.4 用户界面测试

在Web系统开发过程中,需要在三个阶段对WebApp的用户界面进行验证与确认。在需求分析阶段,对界面模型进行评审,确保与利益相关者的需求及分析模型的其他元素相一致;在设计阶段,对界面设计模型进行评审,确保已经达到了为所有用户界面建立的通用质量标准(第15章),并且正确描述了特定于应用的界面设计问题;由于用户交互是通过界面的语法和语义来表示的,因此,在测试阶段,重点转移到特定于应用的用户交互方面的执行。另外,测试提供了对可用性的最终评估。

建议 除了面向WebApp的详细设计说明以外,这里记录的界面策略可应用于所有类型的客户/服务器软件。

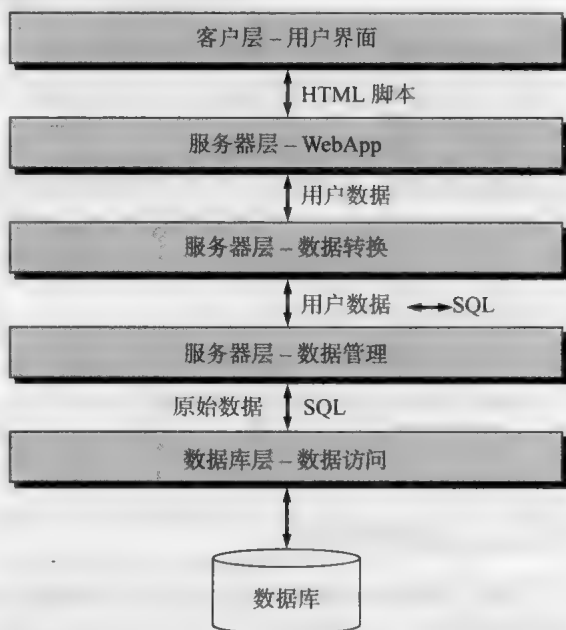


图 25-2 交互层

25.4.1 界面测试策略

界面测试检查用户界面的交互机制,并从美学角度对用户界面进行确认。界面测试的总体测试策略是:(1)发现与特定的界面机制相关的错误(例如,未能正确执行菜单链接的错

^① 数据管理层一般合并了SQL调用层接口(SQL-CLI),如Microsoft OLE/ADO或Java数据库连接(JDBC)。

误，或者输入数据格式的错误)；(2) 发现界面实现导航语义方式的错误、WebApp 的功能性错误或内容显示错误。为了实现此策略，必须启动下面的一些战术步骤：

- 对界面要素进行测试，确保设计规则、美学和相关的可视化内容对用户有效，且没有错误。
- 采用与单元测试类似的方式测试单个界面机制。例如，设计测试用例对所有的表单、客户端脚本、动态 HTML、脚本、流内容及应用的特定界面机制（例如，电子商务应用中的购物车）进行测试。
- 对于特殊的用户类，在用例或导航语义单元（NSU，第 17 章）的环境中测试每一种界面机制。
- 与选择用例及 NSU 有所不同，此方法要对所有界面进行测试，发现界面的语义错误。正是在这个阶段，进行一系列的可用性测试。
- 在多种环境（例如，浏览器）中对界面进行测试，确保其兼容性。

549

25.4.2 测试界面机制

当用户与 WebApp 交互时，是通过一种或多种界面机制来发生交互。在下面的段落中，对每一种界面机制，我们简要介绍一下测试时需要考虑的内容 [Spl01]。

链接。对每个导航链接进行测试，以确保获得了正确的内容对象或功能[⊖]。测试包括与界面布局（例如，菜单条、索引项、每个内容对象的内部链接、外部 WebApp 的链接）相联系的所有链接。

建议 外部链接的测试应该贯穿 WebApp 的整个生命期，支持策略的一部分应该是有规律的、定期的链接测试。

表单。在宏观层次上进行测试，以确保：(1) 对表单中的标识域给出正确标记，并且为用户可视化地标识出强制域；(2) 服务器接收到了表单中包括的所有信息，并且在客户端与服务器之间的传输过程中没有数据丢失；(3) 当用户没有从下拉菜单或按钮组中进行选择时，使用合适的默认项；(4) 浏览器功能（例如，“回退”箭头）没有破坏输入到表单中的数据；(5) 执行对输入数据进行错误检查的脚本工作正常，并且提供了有意义的错误信息。

建议 每当流行的浏览器的新版本发布时，都应该重新进行客户端脚本的测试及与动态 HTML 相关的测试。

在更具有目标性的层次上，测试应该确保：(1) 表单域有合适的宽度和数据类型；(2) 表单建立了合适的安全措施，防止用户输入的文本字符串长度大于某预先定义的最大值；(3) 对下拉菜单中的所有合适的选项进行详细说明，并按照对最终用户有意义的方式排序；(4) 浏览器“自动填充”特性不会导致数据输入错误；(5) Tab 键（或其他键）能够使输入焦点在表单域之间正确移动。

客户端脚本。当脚本运行时，使用黑盒测试发现处理中的一些错误。由于脚本输入通常来自作为表单处理组成部分所提供的的数据，因此这些测试通常与表单测试联合进行。

550

动态 HTML。运行包含动态 HTML 的每一个网页，确保动态显示正确。另外，应该进行兼容性测试，以确保动态 HTML 在支持 WebApp 的环境配置中工作正常。

弹出窗口。进行一系列测试，以确保：(1) 弹出窗口具有合适的大小和位置；(2) 弹出窗口没有覆盖原始的 WebApp 窗口；(3) 弹出窗口的美学设计与界面的美学设计相一致；

⊖ 这些测试可以作为界面测试或导航测试的一部分。

(4) 附加到弹出窗口上的滚动条和其他控制机制被正确定位,并具有所需的功能。

CGI 脚本。一旦已经接收到经过验证的数据,黑盒测试的重点就集中在数据的完整性(当数据被传递给 CGI 脚本)和脚本处理。此外,进行性能测试以确保服务器端的配置符合 CGI 脚本多重调用的处理要求 [Spl01]。

流内容。测试应该证明流数据是最新的、显示正确、能够无错误地暂停,并且很容易重新启动。

cookie :服务器端的测试和客户端的测试都是需要的。在服务器端,测试应该确保一个 cookie 被正确构建(包含正确的数据),并且在请求特定的内容和功能时,此 cookie 能够被正确地传输到客户端。此外,测试此 cookie 是否具有合适的持久性,确保有效日期正确。在客户端,用测试来确定 WebApp 是否将已有的 cookie 正确地附到了特定的请求上(发送给服务器)。

特定应用的界面机制。测试是否与界面机制定义的功能和特性清单相符合。例如, Splaine 和 Jaskiel[Spl01] 为电子商务应用中所定义的购物车功能提出了下面的检查单:

- 对能够放置到购物车中的物品的最小数量和最大数量进行边界测试(第 23 章)。
- 对一个空购物车的“结账”请求进行测试。
- 测试从购物车中正确地删除一件物品。
- 测试一次购买操作是否清空了购物车中的内容。
- 测试购物车内容的持久性(这一点应该作为客户需求的一部分详细说明)。
- 测试 WebApp 在将来某个日期是否能够记起购物车的内容(假设没有购买活动发生)。

551

25.4.3 测试界面语义

一旦对每一个界面功能都已经进行了“单元”测试,就可以将界面测试的关注点转移到界面的语义测试。界面的语义测试是要“评价设计在照顾用户、提供清楚的指导、传递反馈并保持语言与方法的一致性方面做得如何” [Ngu00]。

从头到尾重新考察一下界面设计模型,我们就能够得到前面段落中所蕴涵问题的部分答案。然而,一旦实现了 WebApp,就应该对每个用例场景(针对每一类用户)进行测试。本质上,用例就变成了设计测试序列的输入。测试序列的目的是发现那些妨碍用户获得与用例相关的目标的错误。

25.4.4 可用性测试

可用性测试也评价用户在多大程度上能够与 WebApp 进行有效交互,以及 WebApp 在多大程度上指导用户行为、提供有意义的反馈、并坚持一致的交互方法。从这种意义上说,可用性测试与界面语义测试是相似的(25.4.3 节)。可用性检查和测试不是集中在某交互目标的语义上,而是要确定 WebApp 界面在多大程度上使用户的生活变得轻松[⊖]。

可用性测试可以由测试人员设计,但是测试本身由最终用户进行。可用性测试可能发生在多种不同的抽象级别:(1)对特定的界面机制(例如,

网络资源 可用性测试的有价值的指导可以在 <http://www.keyrlevance.com/articles/usability-tips.htm> 找到。

⊖ 这种问题常用术语“用户友好性”来表示。当然,问题在于一个用户对于“友好”界面的感觉可能根本不同于另一个用户。

表单)的可用性进行评估;(2)对所有网页(包括界面机制、数据对象及相关的功能)的可用性进行评估;(3)考虑整个 WebApp 的可用性。

可用性测试的第一步是确定一组可用性类别,并对每一类可用性建立测试目标。下面的测试类别和目标(以问题的形式书写)举例说明了这种方法^①:

交互性。交互机制(例如,下拉菜单、按钮、指针)容易理解和使用吗?

布局。导航机制、内容和功能放置的方式是否能让用户很快地找到它们?

可读性。文本是否很好地编写,并且是可理解的^②? 图形表示是否容

易理解?

美学。布局、颜色、字体和相关的特性是否使 WebApp 易于使用? 用户对 WebApp 的外观是否“感觉舒适”?

显示特性。WebApp 是否使屏幕的大小和分辨率得到了最佳使用?

时间敏感性。是否能够及时使用或获取重要的要素、功能和内容?

个性化。WebApp 是否能够适应多种用户或个别用户的特殊要求?

可访问性。残疾人是否可以使用该 WebApp?

对于上面每一种可用性,都需要设计一系列测试。在某些情况下,“测试”可以是对网页的可视化审查;而在有些情况下,可以重新执行界面的语义测试,但在下面的实例中,可用性是极为重要的。

作为一个例子,我们考虑对交互和界面机制进行可用性评估。Constantine 和 Lockwood [Con03] 建议应该对下列界面要素进行可用性评审和测试:动画、按钮、颜色、控制、对话、域、表单、框架、图形、标签、链接、菜单、消息、导航、页、选择器、文本和工具条。评估每个要素时,可以由执行测试的用户对其进行定性分级。图 25-3 描述了用户可能选择的一系列评估“级别”。这些级别可以应用于每个单独的要素、所有的网页或者整个 WebApp。



图 25-3 可用性的定性评估

软件工具 Web 用户界面测试工具

[目标] 使用 Web 用户界面测试工具的目的是确定网页或网站中的可用性问题或可访问性问题。

[机制] 这里列出了两种类型的工具。有些工具提示输入 Web 页的 URL, 并遵照一

组启发式, 提供了改正失败的建议列表。当用户在网站的网页上工作时, 有些工具捕捉用户动作, 并且提醒用户。

[代表性工具]^③

- <http://www.usabilla.com/>, usabilla 是一个

① 关于可用性的其他信息参见第 15 章。

② FOG 可读性指数和其他工具可以用来定量地评估可读性。更多的细节参见 <http://developer.gnome.org/gdp-style-guide/stable/usability-readability.html>。

③ 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

在线工具，它允许开发者跟踪用户的动作，并在使用网页期间收集用户的意见。

- <http://www.google.com/analytics/>，Google Analytics 是一个在线工具，可提供综合的网站数据跟踪和分析工具，用于评估网站的可用性。
- <http://valet.webthing.com/access/url.html>，Web Valet 提供了在线服务，检查

网页的可访问性问题。

- <http://wave.webaim.org/>，Wave 提供了在线服务，通过对网页进行标记来说明可访问性问题。
- <http://www.sidar.org/hera/index.php.en>，Hera 提供了一个在线服务，运用 Web 内容访问指南来检查网页的可访问性问题。

25.4.5 兼容性测试

不同的计算机、显示设备、操作系统、浏览器和网络连接速度都会对 WebApp 的运行造成很大的影响。每一种计算配置都可能使客户端的处理速度、显示分辨率和连接速度有所不同。操作系统反复无常的行为也可能导致 WebApp 的处理问题。不管 WebApp 中 HTML 的标准化程度如何，不同的浏览器有时会产生稍微不同的结果。特殊的配置所需要的插件可能容易获得，也可能不容易获得。兼容性测试试图在 WebApp 上线前发现这些问题。

兼容性测试的第一步是定义一组“通常遇到”的客户端计算配置和它们的变体。实际做法是创建一种树结构，并在上面标识每一种计算平台、典型的显示设备、此平台支持的操作系统、可用的浏览器、可靠的 Internet 连接速度及类似信息。下一步，导出一系列兼容性确认测试，可以从现有的界面测试、导航测试、性能测试和安全性测试中导出。这些测试的目的是发现由于配置差异所导致的错误和运行问题。

554

SafeHome WebApp 测试

[场景] Doug Miller 的办公室。

[人物] Doug Miller，SafeHome 软件团队经理；Vinod Raman，SafeHome 产品软件团队工程的成员。

[对话]

Doug: 对于 SafeHomeAssured.com 电子商务 WebApp 0.0 版，你是怎样看的？

Vinod: 外包供应商已经做了很好的工作。Sharon（供应商的开发经理）告诉我，他们正在按我们说的进行测试。

Doug: 我希望你和团队的其他人能够对电子商务网站做一点非正式的测试。

Vinod（作苦相）: 我想我们将雇用第三方测试公司对 WebApp 进行确认测试。我们仍然在致力于推出产品软件。

Doug: 我们将雇用测试供应商进行性能测

试和安全性测试，并且我们的外包供应商已经在进行测试了。只是想从另外一种角度看看是否会有帮助，况且，我们想控制成本，所以……

Vinod（叹息）: 你在期待什么？

Doug: 我想确信界面和所有的导航都是可靠的。

Vinod: 我想我们可以从每个主要界面功能的用例开始。

学习 SafeHome

详细说明你需要的 SafeHome 系统

购买一套 SafeHome 系统

取得技术支持

Doug: 很好。但是，要走通所有的导航路径，才能得出结论。

Vinod（浏览记录用例的笔记本）: 是的，

当你选择“详细说明你需要的 SafeHome 系统”时，此系统将使你：

选择 SafeHome 构件

获得 SafeHome 构件建议

我们要当心每一条路径的语义。

Doug：测试时，需要检查出现在每一个导航节点的内容。

Vinod：当然，还有功能元素。谁在测试

可用性？

Doug：哦，测试供应商将配合可用性测试。我们已经雇了一个市场调查公司列出 20 个进行可用性研究的典型用户，但是，如果你发现了任何可用性问题……

Vinod：我会转给他们。

Doug：谢谢！Vinod。

25.5 构件级测试

构件级测试也称功能测试，它集中于一系列的测试，试图发现 WebApp 功能方面的错误。每个 WebApp 功能都是一个软件模块（用多种编程语言或脚本语言中的一种实现的），并且可以用第 23 章讨论的黑盒（及在某些情况下的白盒）技术对其进行测试。

构件级测试用例通常受表单级的输入驱动。一旦定义了表单数据，用户就可以选择按钮或其他控制机制来启动运行。在测试基于表单的输入和应用于表单的功能方面，适合采用等价类划分、边界值分析和路径测试（第 23 章）。

除了这些测试用例设计方法，还可以使用称为强制错误测试（forced error testing）[Ngu01] 的技术导出测试用例，这些测试用例故意使 WebApp 构件进入错误条件，目的是发现错误处理过程中发生的错误（例如，不正确或不存在的错误提示信息，由于错误的发生导致 WebApp 失败，由错误的输入而导致的错误输出，与构件处理有关的副作用）。

555

每个构件级测试用例详细说明了所有的输入值和由构件提供的预期输出。将测试过程中产生的实际输出数据记录下来，以供将来的支持和维护阶段参考。

25.6 导航测试

用户在 WebApp 中旅行的过程与访问者在商店或博物馆中漫步的过程很相似。可走很多路径，可以有很多站，可学习和观看很多事情，启动很多活动，并且可以做决策。如我们所讨论的那样，每个访问者到来时都有一系列的目标，在这个意义上，导航过程是可预测的。同时，导航过程又可能是无法预测的，因为访问者受到他所看到的或学到的某件事的影响，可能选择一条路径或启动一个动作，而这对于最初的目标并不是典型的路径或动作。导航测试的工作是：（1）确保允许 WebApp 用户经由 WebApp 游历的机制都是功能性的；（2）确认每个导航语义单元（NSU）都能够被合适的用户类获得。

25.6.1 测试导航语法

实际上，导航测试的第一个阶段在界面测试期间就开始了。应对导航功能进行测试，以确保每个导航都执行了预计的功能。Splaine 和 Jaskiel[Spl01] 建议应该对下面的每个导航功能进行测试：链接及所有类型的锚、重定向（当用户请求一个不存在的 URL 时）、书签、框架和框架集、站点地图以及内部搜索引擎。

引述 我们没有迷路，我们面临定位挑战。

John M. Ford

前面已经提到的某些测试可以由自动工具执行（例如，链接检查），而另外一些要手工设计和执行。导航测试的目的始终是确保在 WebApp 上线之前发现导航功能方面的错误。

25.6.2 测试导航语义

在第 17 章，我们将导航语义单元（NSU）定义为“一组信息和相关的导航结构，在完成相关的用户需求的子集时，这些导航结构会相互协作”[Cac02]。每个 NSU 由一系列连接导航节点（例如，网页、内容对象或功能）的导航路径（称为“导航的路”）定义。作为一个整体，每个 NSU 允许用户获得特殊的需求，这种特殊的需求是针对某类用户，由一个或多个用例定义的。导航测试应检查每个 NSU，以确保能够获得这些需求。

在测试每个 NSU 时，Web 工程团队一定要回答下面的问题：

- 此 NSU 是否没有错误地全部完成了？
- 在为此 NSU 定义的导航路径的上下文中，（为一个 NSU 定义的）每个导航节点是否都是可达的？
- 如果使用多条导航路径都能完成此 NSU，每条相关的路径是否都已经被测试？
- 如果使用用户界面提供的指导来帮助导航，当导航进行时，它们方向正确并可理解吗？
- 是否具有返回到前一个导航节点及导航路径开始位置的机制（不同于浏览器的“回退”箭头）？
- 大型导航节点（即一个长的网页）中的导航机制工作正常吗？
- 如果一个功能在一个节点上运行，并且用户选择不提供输入，那么 NSU 的剩余部分能完成吗？
- 如果一个功能在一个节点上运行，并且在功能处理时发生了一个错误，那么 NSU 能完成吗？
- 在到达所有节点之前，是否有办法终止导航，然后又能返回到导航被终止的地方并从那里继续？
- 从站点地图可以到达每个节点吗？节点的名字对最终用户有意义吗？
- 如果可以从某外部的信息源到达 NSU 中的一个节点，那么有可能推移到导航路径的下一个节点吗？有可能返回到导航路径的前一个节点吗？
- 运行 NSU 时，用户知道他在内容体系结构中的位置吗？

提问 我们测试 NSU 时，必须提出和回答哪些问题？

建议 如果我们在 Web 工程的分析和设计中没有创建 NSU，则可以将用例应用于导航测试用例的设计，需要提出和回答的一系列问题是一样的。

如同界面测试和可用性测试，导航测试应该由尽可能多的不同的支持者进行。测试的早期阶段由 Web 工程师进行，但后来的测试应该由其他的项目利益相关者、独立的测试团队进行，最后应该由非技术用户进行，目的是彻底检查 WebApp 导航。

软件工具

Web 导航测试工具

[目标] 使用 Web 导航测试工具的目标是识别网站中不可达的任何毁坏的链接或网页。

[机制] 这些工具提示输入 Web 页的 URL，

并扫描其标记语言，查找不能返回正确网页类型的链接。某些工具试图抓取整个网站来寻找深层链接中的错误。

[代表性工具][⊖]

- <http://validator.w3.org/checklink/>，一个在线的 WC3 链接检查工具，它可分析 HTML 和 XHTML 文档中无效的链接。

- <http://www.relsoftware.com/>，Rel Link Checker Lite 的下载网站，Rel Link Checker Lite 是一个免费工具，可用于确认无效链接和孤立文件。

557

25.7 配置测试

配置的可变性和不稳定性是使 Web 工程面临挑战的重要因素。硬件、操作系统、浏览器、存储容量、网络通信速度和多种其他的客户端因素对每个用户都是难以预料的。另外，某个用户的配置可能会有规律地改变（例如，操作系统升级、新的 ISP 和连接速度），其结果可能是客户端环境容易出错，这些错误既微妙又重要。如果两个用户不是在相同的客户端配置中工作，那么一个用户对 WebApp 的印象及与 WebApp 的交互方式可能与另一个用户的体验有很大不同。

配置测试的工作不是去检查每一个可能的客户端配置，而是测试一组很可能的客户端和服务器端配置，确保用户在所有配置中的体验都是一样的，并且将特定于特殊配置的错误分离出来。

25.7.1 服务器端问题

在服务器端，以配置测试用例验证所计划的服务器配置（即 WebApp 服务器、数据库服务器、操作系统、防火墙软件、并发应用）能够支持 WebApp，而不会发生错误。

当设计服务器端的配置测试时，Web 工程师应该考虑服务器配置的每个构件。在服务器端的配置测试期间，需要提出及回答以下问题：

- WebApp 与服务器操作系统完全兼容吗？
- 当 WebApp 运行时，系统文件、目录和相关的系统数据是否正确创建？
- 系统安全措施（例如，防火墙或加密）允许 WebApp 运行并对用户提供服务，而不发生冲突或性能下降吗？
- 是否已经利用所选择的分布式服务器配置[⊖]（如果存在一种配置）对 WebApp 进行了测试？
- 此 WebApp 是否与数据库软件进行了适当的集成？是否对数据库的不同版本敏感？
- 服务器端的 WebApp 脚本运行正常吗？
- 系统管理员错误对 WebApp 运行的影响是否已经得到检查？
- 如果使用了代理服务器，在站点测试时，是否已经明确这些代理服务器在配置方面的差异？

提问 进行服务器端配置测试时，必须提出和回答哪些问题？

25.7.2 客户端问题

在客户端，配置测试更多地集中在 WebApp 与配置的兼容性上，这些配置包括下面构件

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

⊖ 例如，可能使用单独的应用服务器和数据库服务器，两台机器之间通过网络连接进行通信。

558

的一种或多种改变 [Ngu01]：硬件、操作系统、浏览器软件、用户界面构件、插件及连通服务（例如，电缆、DSL、WiFi）。除了这些构件，其他配置变量包括网络软件、ISP 的难以预测的变化及并发运行的应用。

为了设计客户端配置测试，Web 工程团队必须将配置变量的数量减少到可管理的数目^①。为了实现这一点，要对每类用户进行评估，以确定此类用户可能遇到的配置。此外，工业市场上的共享数据可以用来预测最可能的构件组合，然后，在这些环境中测试 WebApp。

软件工具 Web 配置测试工具

[目标] 使用 Web 配置测试工具的目标是确定当用不同的 Web 浏览器和操作系统组合显示页面时可能会出现的问题。

[机制] 这些工具提示输入 Web 页的 URL，并允许从很多浏览器和操作系统组合中进行选择。此类工具将显示网页的缩略图，就如同在所选择的每种浏览器版本中所显示的那样。

[代表性工具]^②

- <http://browsershots.org/>, Browsershots 提供在线服务，可以利用很多不同的浏览器和操作系统对网站进行测试。
- <http://testingbot.com/>, TestingBot 提供了有限免费试用的在线服务，允许使用很多不同的浏览器和操作系统对网站进行测试。

25.8 安全性测试^③

WebApp 的安全性测试是一个复杂的主题，在有效地完成安全性测试之前，必须要对该主题有充分的了解^④。WebApp 和其所处的客户端和服务端环境对于一些人来说是很有吸引力的攻击目标，这些人包括：外部的电脑黑客，对单位不满的员工，不诚实的竞争者，以及其他想偷窃敏感信息、恶意修改内容、降低性能、破坏功能或者给个人、组织或业务制造麻烦的人。

安全性测试用于探查在某些方面所存在的弱点，比如客户端环境，以及当数据从客户端传到服务器并从服务器再传回客户端时所发生的网络通信及服务器端环境。这些领域中的每一个都可能会受到攻击。发现可能会被怀有恶意的人利用的弱点，这是安全性测试人员的任务。

在客户端，弱点通常可以追溯到早已存在于浏览器、电子邮件程序或通信软件中的缺陷。在服务器端，薄弱环节包括拒绝服务攻击和恶意脚本，这些恶意脚本可以被传到客户端，或者用来使服务器操作丧失能力。另外，服务器端数据库能够在没有授权的情况下被访问（数据窃取）。

引述 对于管理业务和存储资产来说，Internet 是一个危险的地方。电脑黑客、解密高手、窥探者、骗子、小偷、故意破坏者、病毒发布者和无赖程序承办商都可以自由行动。

Dorothy,
Peter Denning

建议 如果 WebApp 是业务关键的，用来维护敏感的数据，或者很可能成为电脑黑客的目标，则将安全性测试外包给擅长于此的供应商是一个好主意。

① 在每一种可能的配置构件的组合中运行测试是非常耗费时间的。

② 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。

在大多数情况下，工具名称被各自的开发者注册为商标。

③ 在第 27 章中，安全性测试还被作为安全性工程的一部分进行讨论。

④ 由 Cross 和 Fisher[Cro07]、Andrew 和 Whittaker[And06] 及 Trivedi[Tre03] 编写的书籍提供了关于此主题的有关信息。

为了防止这些（和很多其他）攻击，可以使用防火墙（firewalls）、鉴定（authentication）、加密（encryption）和授权（authorization）技术。应该设计安全性测试，探查每种安全性技术来发现安全漏洞。

在设计安全性测试时，需要深入了解每一种安全机制的内部工作情况，并充分理解所有网络技术。在很多情况下，应将安全性测试外包给擅长这些技术的公司。

关键点 应该将安全性测试设计为检验防火墙、鉴定、加密和授权。

软件工具 Web 安全性测试工具

[目标] Web 安全性测试工具的目标是帮助识别网站中可能存在的安全性问题。

[机制] 这些工具通常可以下载，并在开发环境中运行。它们检查应用，查找可以注入可能改变网站功能的有害数据脚本。有些工具可以作为探查工具来使用。

[代表性工具]^①

- <http://www.mavitunasecurity.com/communityedition/>，工具 Netsparker 的

下载网站，此工具检查 WebApp 是否存在 SQL 注入漏洞。

- <http://enyojs.com/>，自由工具 N-Stalker 的下载站点，该工具使用网络攻击特征库对网站执行很多安全性检查。
- <http://code.google.com/p/skipfish/>，skipfish 工具的下载站点，此工具通过抓取网站的页面给出安全漏洞报告。

25.9 性能测试

你的 WebApp 要花好几分钟下载内容，而竞争者的站点下载相似的内容只需几秒钟，没有什么比这更让人感到不安了；你正设法登录到一个 WebApp，却收到“服务器忙”的信息，建议你过一会儿再试，没有什么比这更让人感到烦恼了；WebApp 对某些情形能够立即做出反应，而对某些情形却似乎进入了一种无限等待状态，没有什么比这更让人感到惊慌了。所有这些事件每天都在 Web 上发生，并且所有这些都是与性能相关的。

560

使用性能测试来发现性能问题，这些问题可能是由以下原因产生的：服务器端资源缺乏、不合适的网络带宽、不适当的数据库容量、不完善或不强大的操作系统能力、设计糟糕的 WebApp 功能以及可能导致客户 - 服务器性能下降的其他硬件或软件问题。性能测试的目的是双重的：（1）了解系统如何对负载（即用户的数量、事务的数量或总的数量）做出反应；（2）收集度量数据，这些数据将促使修改设计，从而使性能得到改善。

25.9.1 性能测试的目标

设计性能测试来模拟现实世界的负载情形。随着同时访问 WebApp 用户数量的增加，在线事务数量或数据量（下载或上载）也随之增加，性能测试将帮助回答下面的问题：

- 服务器响应时间是否降到了值得注意的或不可接受的程度？
- 在什么情况下（就用户、事务或数据负载来说），性能变得不可接受？

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

- 哪些系统构件应对性能下降负责?
- 在多种负载条件下, 对用户的平均响应时间是多少?
- 性能下降是否影响系统的安全性?
- 当系统的负载增加时, WebApp 的可靠性和准确性是否会受影响?
- 当负载大于服务器容量的最大值时会发生什么情况?
- 性能下降是否对公司的收益有影响?

为了得到这些问题的答案, 要进行两种不同的性能测试: (1) 负载测试, 在多种负载级别和多种组合下, 对真实世界的负载进行测试。(2) 压力测试, 将负载增加到强度极限, 以此来确定 WebApp 环境能够处理的容量。在下面的两节中将分别考虑每种测试的策略。

561

25.9.2 负载测试

负载测试的目的是确定 WebApp 和其服务器环境如何响应不同的负载条件。在进行测试时, 下面变量的排列定义了一组测试条件:

N , 并发用户的数量

T , 每单位时间的在线事务数量

D , 每次事务服务器处理的数据负载

每种情况下, 在系统正常的操作范围内定义这些变量。当每种测试条件运行时, 收集下面的一种或多种测量数据: 平均用户响应时间, 下载标准数据单元的平均时间, 或者处理一个事务的平均时间。Web 工程团队对这些测量进行检查, 以确定性能的急剧下降是否与 N 、 T 和 D 的特殊组合有关。

负载测试也可以用于为 WebApp 用户估计建议的连接速度。以下面的方式计算总的吞吐量 P :

$$P = N \times T \times D$$

例如, 考虑一个大众体育新闻站点。在某一给定的时刻, 2 万个并发用户平均每两分钟提交一次请求 (事务 T)。每一次事务需要 WebApp 下载一篇平均长度为 3KB 的新文章, 因此, 可如下计算吞吐量:

$$P = (20\,000 \times 0.5 \times 3\text{KB}) / 60 = 500\text{KB/s} = 4\text{Mb/s}$$

因此, 服务器的网络连接将不得不支持这种数据传输速度, 应对其进行测试, 以确保它能够达到所需要的数据传输速度。

25.9.3 压力测试

压力测试是负载测试的继续, 但是, 在压力测试中, 我们强迫变量 N 、 T 和 D 满足操作极限, 然后超过操作极限。这些测试的目的是回答下面的问题:

- 系统“逐渐”降级吗? 或者, 当容量超出时, 服务器停机吗?
- 服务器软件会给出“服务器不可用”的提示信息吗? 更一般地说, 用户知道他们不能访问服务器吗?
- 服务器队列请求增加资源吗? 一旦容量要求减少, 会释放队列所占用的资源吗?

建议 至少在被最终用户察觉到问题前, WebApp 性能的很多方面是难于测试的, 包括网络负载、网络接口硬件的反复无常的行为及类似的问题。

建议 如果一个 WebApp 使用多个服务器提供巨大容量, 则必须在多服务器环境中进行负载测试。

关键点 压力测试的目的是更好地理解: 当系统所承受的压力超过它的操作极限时, 系统是如何失效的?

- 当容量超过时事务会丢失吗?
- 当容量超过时数据完整性会受到影响吗?
- N 、 T 和 D 的哪些值迫使服务器环境失效? 如何来证明失效? 自动通知会被发送到位于服务器站点的技术支持人员那里吗?
- 如果系统失效, 需要多长时间才能回到在线状态?
- 当容量达到 80% 或 90% 时, 某些 WebApp 功能 (例如, 计算密集的功能、数据流动能力) 会被停止吗?

有时将压力测试的变型称为脉冲 / 回弹测试 (spike/bounce testing) [Spl01]。在这种测试中, 增加负载以达到最大容量, 然后迅速回落到正常的操作条件, 然后再增加。通过回弹系统负载, 测试者能够确定服务器如何调度资源来满足非常高的需求, 然后当一般条件再现时释放资源 (以便为下一次脉冲做好准备)。

软件工具 Web 性能测试工具

[目标] Web 性能测试工具的目标是查找使系统性能低下的瓶颈或者模拟可能导致网站完全失败的条件。

[机制] 在线工具提示输入 Web 资源的 URL 地址。某些工具自动构造一系列模拟的负载测试。某些工具收集网页负载的统计数据以及导航网站时服务器的响应时间。

[代表性工具]^①

- <http://loadimpact.com/>, LoadImpact 是一个在线工具, 可以在 Web 服务器上使用模拟的用户负载进行负载影响测试。
- <http://www.websitepulse.com/help/testtools.website-test.html/>, WebSitePulse 是一个在线工具, 可以测量服务器的可用性和

网站的响应时间。

- <http://www.websiteoptimization.com/services/analyze/>, Web Page Analyzer 是一个在线工具, 可以测量网站的性能, 并提供一个建议的更新列表来改进装载次数。
- <http://developer.yahoo.com/yslow/>, yslow 是一个在线工具, 可以基于高性能网站的开发规则对网页进行分析, 并给出改进建议。
- <http://tools.pingdom.com/ftp/>, pingdom 是一个在线工具, 它通过单独分析构件元素, 对网页装载时间的瓶颈进行测量。

25.10 小结

WebApp 测试的目标是对每一种 WebApp 质量维度进行检查, 发现可能导致质量失效的错误或问题。应该对内容、功能、结构、可用性、导航性、性能、兼容性、互操作性、容量和安全性方面进行重点测试, 在 WebApp 设计完成后进行评审, 一旦实现了 WebApp, 就对其进行测试。

WebApp 测试策略检查每个质量维度, 从考察内容、功能或导航“单元”开始。一旦单独的单元都已经被确认, 重点就转到测试整个 WebApp。为了完成这项工作, 很多测试来自

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

于用户的看法，并由用例所包含的信息所驱动。应编写 WebApp 测试计划，并确定测试步骤、工作产品（例如，测试用例）以及测试结果的评价机制。测试过程包括 7 个不同的测试类型。

内容测试（和评审）主要对内容的多种分类进行测试，目的是发现语义或语法上的错误，这些错误会影响内容的准确性，或对展示给最终用户的方式有影响。界面测试检查用户与 WebApp 之间通信的交互机制，并对界面的美学方面进行确认。目的是发现由于实现糟糕的交互机制、遗漏、不一致或界面语义不明确所导致的错误。

导航测试使用从建模活动中得出的用例，在测试用例的设计中，测试用例对照导航设计检查每个使用场景。对导航机制进行测试，以确保识别并改正妨碍用例完成的任何错误。构件测试检查 WebApp 中的内容和功能单元。

配置测试试图发现针对特殊的客户或服务器环境的错误和兼容性问题，然后进行测试，发现与每种可能配置有关的错误。安全性测试包括一系列测试，探测 WebApp 及其环境中存在的弱点，目的是发现安全漏洞。性能测试包括一系列测试，当对服务器端资源容量的要求增加时，评估 WebApp 的响应时间及可靠性。

习题与思考题

- 25.1 是否存在应该完全忽视 WebApp 测试的任何情况？
- 25.2 用你自己的话讨论在 WebApp 环境下测试的目标。
- 25.3 兼容性是一项重要的质量维度，必须对哪方面进行测试才能确保 WebApp 具有兼容性？
- 25.4 哪些错误趋向于更加严重，是客户端错误还是服务器端错误？为什么？
- 25.5 WebApp 的哪些元素可以进行“单元测试”？哪些类型的测试只能在 WebApp 元素被集成之后进行？
- 25.6 开发正式的书面测试计划是否总是必要的？给出解释。
- 25.7 是否可以公平地说，总的 WebApp 测试策略开始于用户可见的元素，然后移向技术元素？这种策略存在例外吗？
- 25.8 内容测试是传统意义上的真正测试吗？给出解释。
- 25.9 描述与 WebApp 的数据库测试有关的步骤。数据库测试是否对客户端和服务端活动有影响？
- 25.10 与界面机制相关的测试和界面语义测试之间的区别是什么？
- 25.11 假设你正在开发满足老年人需要的在线药房（YourCornerPharmacy.com）。药房提供了典型的功能，但也为每位客户维护一个数据库，使得它可以提供药品信息和潜在的药品相互作用警告。讨论针对此 WebApp 的任何特殊的可用性测试。
- 25.12 假设你已经为 YourCornerPharmacy.com（习题 25.11）实现了一个药品相互作用检查功能。讨论必须进行的构件级测试的类型，以确保这项功能工作正常。（注意：实现这项功能将必须使用数据库。）
- 25.13 导航语法测试和导航语义测试之间的区别是什么？
- 25.14 测试 WebApp 可能在服务器端遇到的每一种配置，这样做可能吗？在客户端呢？如果不可能，Web 工程师如何选择有意义的配置测试集合？
- 25.15 安全性测试的目标是什么？谁来进行这种测试活动？
- 25.16 YourCornerPharmacy.com（习题 25.11）已经取得了广泛成功，在运行的前两个月里用户数量剧增。对于固定的服务器端资源集合，画一张图，将可能的响应时间描述为用户数量的函数。对图进行标注，指出“响应曲线”的兴趣点。
- 25.17 为使应用成功，CornerPharmacy.com（习题 25.11）已经实现了一个特殊的服务——单独处理

处方的重新填写。平均情况下, 1000 个并发用户每两分钟提交一次重填请求, WebApp 下载 500B 的数据块来响应。此服务器需要具有的吞吐量是多少 Mb/s?

25.18 负载测试和压力测试之间的区别是什么?

扩展阅读与信息资源

WebApp 测试方面的文献仍在不断发表。这方面最新出版的书籍中, 由 Andrews 和 Whittaker (《How to Break Web Software》, Addison-Wesley, 2006)、Ash (《The Web Testing Companion》, Wiley, 2003)、Nguyen 和他的同事 (《Testing Applications for the Web》, second edition, Wiley, 2003)、Dustin 和他的同事 (《Quality Web Systems》, Addison-Wesley, 2002) 以及 Splaine 和 Jaskiel[SPL01] 撰写的书论述得最全面。Whitaker 和他的同事 (《How Google Tests Software》, Addison-Wesley, 2012) 描述了另外的 Web 测试实践。Mosley (《Client-Server Software Testing on the Desktop and the Web》, Prentice Hall, 1999) 论述了客户端和服务器的测试问题。

David (《Selenium 2 Testing Tools: A Beginner's Guide》, Packit Publishing, 2012) 和 Stottlemeyer (《Automated Web Testing Toolkit》, Wiley, 2001) 介绍了 WebApp 测试策略和方法方面的有用信息, 并对自动化测试工具进行了有价值的讨论。Graham 和他的同事 (《Experiences of Test Automation》, Addison-Wesley, 2012 和《Software Test Automation》, Addison-Wesley, 1999) 介绍了自动化工具方面的其他资料。

微软 (《Performance Testing Guidance for Web Applications》, Microsoft Press, 2008) 和 Subraya (《Integrated Approach to Web Performance Testing》, IRM Press, 2006) 对 WebApp 性能测试进行了详细论述。Hope 和 Walther (《Web Security Testing Cookbook》, O'Reilly, 2008)、Skoudis (《Counter Hack Reloaded》, second edition, Prentice Hall, 2006)、Andreu (《Profession Pen Testing for Web Applications》, Wros, 2006)、Chirillo (《Hack Attacks Revealed》, second edition, Wiley, 2003)、Splaine (《Testing Web Security》, Wiley, 2002) 以及 Klevisky 和他的同事 (《Hack I.T.: Security through Penetration Testing》, Addison-Wesley, 2002) 为必须设计安全性测试的人提供了非常有用的信息。另外, 讲述软件安全测试的书籍通常能够为那些必须测试 WebApp 的人提供重要的指导。有代表性的书籍包括: Engebretson (《Basics of Hacking and Penetration Testing》, Syngress, 2011)、Basta 和 Halton (《Computer Security and Penetration Testing》, Thomson Delmar Learning, 2007)、Wysopal 和他的同事 (《The Art of Software Security Testing》, Addison-Wesley, 2006) 以及 Gallagher 和他的同事 (《Hunting Security Bugs》, Microsoft Press, 2006)。

在网上可以获得大量 WebApp 测试的信息资源, 最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

测试移动 App

要点浏览

概念：移动 App 测试相关活动的唯一目标是揭示移动 App 在内容、功能、易用性、导航性、性能、容量和安全性等方面的错误。为了做到这一点，就要采用评审和执行测试的测试策略。

人员：软件工程师和其他的项目利益相关者（经理、客户和最终用户）都应参加移动 App 的测试。

重要性：如果移动 App 的用户遭遇到错误或困难，他们便会到别处寻求所需的个性化的内容和功能。因此，在将移动 App 作为产品投放到 App 销售商或存入仓库之前，必须努力寻找和纠正尽可能多的错误。

步骤：移动 App 的测试过程要从关注移动

App 用户的可见方面开始，一直进行到有关所用技术和基础架构的测试。所执行的测试步骤包括：内容测试、界面测试、导航测试、构件测试、配置测试、性能测试和安全性测试。

工作产品：往往首先要制定移动 App 的测试计划。为每个测试步骤开发一套测试用例，并且保存测试结果以供将来使用。

质量保证措施：虽然无法确定已经完成了每一个所要做的测试，但可以肯定测试已发现了错误（并且已纠正了那些错误）。此外，如果已经制定了测试计划，便可以进行检查，以确保所有计划的测试均已完成。

推动着 WebApp 项目的紧迫感也波及到移动 App 项目之中。利益相关者都担心会错过营销机会，纷纷催促上架移动 App。然而诸如性能测试和安全性测试这样的一些技术活动往往被安排在开发过程的后期阶段，因而常常是匆忙地结束。本应在测试阶段实施的可用性测试却被推迟到交付之前进行。这样就会导致灾难性错误。为避免出现这种情况，开发团队成员必须确保每一项工作产品都具有高质量，正如 Soasta 在他发表的白皮书 [Soa11] 中所表述的：

移动技术的发展简直是直上云霄，其速度超过以往的任何技术，这是史无前例的最快创新采用曲线。而且它对于业务模式具有重要意义。你的产品会快速进入市场，而且要准备被迅速采用，如果你的应用系统性能不佳，或是承载时发生故障，那么正准备取代你的市场地位的很多竞争对手就会使您陷入困境。

关键概念

自动化
检查单
准则
国际化
实时测试
策略
压力测试
自然环境测试
测试矩阵
工具与环境
可用性测试

使用可执行的测试用例无法单独测试移动 App 的需求和设计模型。你和你的团队应该进行技术评审（第 20 章），并测试可用性（第 15 章）及性能。作为验证移动 App 能力的一种机制，必须考虑环境的情况，此时配置测试显得尤为重要。在将移动 App 交付给最终用

户之前，这样做的目的是发现和纠正错误。

在制定移动 App 的测试策略时，要问的几个重要问题是 [Scho09]：

- 在你和用户一起测试之前，是否必须建立功能齐全的原型？
- 你应该利用用户设备进行测试还是提供测试设备？
- 在测试中你应该拥有什么设备和用户组？
- 实验室测试和远程测试相比各有什么优缺点？

在本章的整个讲述中，我们会涉及以上每个问题。

26.1 测试准则

完全在移动设备上运行的移动 App 既可以使用传统的软件测试方法（第 23 章）进行测试，也可以在个人计算机上使用模拟运行的方式测试。另一方面，利用基于服务器资源的瘦客户端移动 App 对测试特别具有挑战性。除了 WebApp 提出的许多测试挑战（第 25 章）以外，瘦客户端移动 App 的测试还必须考虑互联网网关和电话网络进行的数据传输 [Was10]。用户期望了解移动 App 运行环境，并提交基于设备的物理位置，以及与可用的网络功能相关的个性化用户体验。但是，使用每个可能的设备和网络配置在一个动态的特定网络环境中测试移动 App，这如果不是不可能也是十分困难的。

要了解移动 App 测试的目标，就应该首先考虑 App 设计师所面临的许多独特的挑战。人们期望所得到的移动 App 具有复杂的功能，并且还具有基于个人计算机的 App 所具有的可靠性。但是移动 App 常驻于具有相对有限资源的移动平台。下列准则为移动 App 的测试提供了基础 [Kea07]：

建议 尽早让用户参与测试活动，这样可以尽早得到有助于纠正设计的反馈信息。

- 在测试以确定瓶颈之前了解网络和设备的运行环境。跨边界测试在 26.4 节讨论。
- 在不受控的现实环境的测试条件下进行测试（基于现场的测试），特别是针对多端网络（multitier）移动 App 进行测试。生产环境中的测试在 26.2.5 节讨论。
- 选择适当的自动测试工具。理想情况下，这些工具应该支持所有要使用的平台，允许测试各种各样类型的屏幕、分辨率和输入机制，如触控式和小键盘等，并实现与外部系统的联通以进行终端对终端的测试。有关移动 App 测试工具更详细的讨论见 26.6 节。
- 利用加载设备平台矩阵法确定最为关键的硬件 / 平台测试组合。特别是在硬件 / 平台组合数较多而测试时间又很短时，这种方法是非常有效的。采用这一方法的详细情况可参看 26.2.3 节。
- 至少检查一次在所有可能的平台上的端对端功能流。当涉及网页服务时，如果没有性能测试工具，跟踪所需的实际网络路径以交付移动 App 的功能是很难的。关于工具的使用在 26.6 节讨论。
- 使用实际设备进行性能测试、图形用户界面（GUI）测试和兼容性测试。虽然可以利用模拟器完成这些测试，但在此仍然建议使用实际设备。用户交互测试在 26.3 节讨论，而性能测试问题在 26.2 节讨论。
- 测量性能只在无线通信和用户负载的实际条件下进行。移动 App 的实时测试问题在 26.5 节讨论。

26.2 测试策略

仅使用技术手段不能保证移动 App 在商业上的成功。如果这些 App 在运行中失灵,或是未能达到预期的使用效果,那么用户将会很快遗弃这些 App。回顾测试工作所具有的两个重要目标是很有意义的,那就是:(1)为在开发活动早期阶段发现缺陷而制定测试用例;(2)验证其具有重要的质量属性。移动 App 应具有的质量属性是基于国际标准 ISO 9126[Spr04]提出的那套软件产品的质量属性,包括功能性、可靠性、易用性、效率、维护性和可移植性(第 19 章)。

制定移动 App 测试策略既要了解软件测试知识,又需理解移动设备及其网络基础架构特性所面临的挑战[Kho12a]。除了具有常规软件测试方法的全面知识(第 22、23 章)以外,移动 App 的测试人员还应该对电信原理有很好的理解,并且要认识到移动操作系统平台的差异和功能。对于这些基础知识,必须要有另外的知识对其加以补充,包括:对不同类型移动测试的深入理解(例如,移动 App 测试、移动手持终端设备的测试、移动网站测试),模拟器的使用,测试自动化工具以及远程数据存取服务(RDA)。上述专题都将在本章后面的小节中陆续讨论。

引述 如果我还
没有死去,那么
我宁愿葬身于手
机之中。

Amanda Holden

569

信息栏 移动 App 测试——检查单

移动 App 咨询有限公司 Appsparq 的首席执行官 Nari Kannan 对测试移动 App 发表了以下建议[Kan11]:

- **概念测试**——在开始开发之前,从移动 App 的未来用户处获取输入信息,有助于确保 App 已囊括必要的系统特征,而不致遗漏最为重要的特征(收集需求和确认技术在第 9 章中讨论)。
- **单元测试和系统测试**——实施正规的单元测试和系统测试针对的是含有多个构件的任何软件应用,并与网络交互作用(第 22、26 章)。
- **用户体验测试**——如在开发过程早期就吸收实际用户参与,便能保证所提供的体验达到利益相关者对易用性和可访问性的期望。移动 App 的易用性测试在 26.3 节讨论。
- **稳定性测试**——确保移动 App 不会由于与

网络或网站服务不兼容而崩溃。在生产环境中测试移动 App 在 26.2.5 节讨论。

- **连接性测试**——测试直接进入所有重要构件和网络资源,这对于确保移动 App 在运行环境中适当地使用是非常重要的。跨界测试在本章 26.6 节讨论。
- **性能测试**——测试移动 App 的能力以满足其非功能性需求(下载时间、处理器速度、存储容量等)。性能测试在 26.2.4 节和 26.5 节讨论。
- **设备兼容性测试**——验证移动 App 是否在所有目标设备上正常运行。该项任务在 26.2.3 节讨论。
- **安全性测试**——确保移动 App 满足其利益相关者设定的安全性需求。安全性保证测试在第 27 章讨论。
- **认证测试**——检查移动 App 是否满足由分发机构制定的标准。

26.2.1 传统方法适用吗

综合性移动 App 测试程序包括第 22 章讨论的通用螺旋方式,以及第 23 ~ 25 章中提到

的客户-服务器架构、实时计算、图形用户界面、WebApp 以及面向对象系统。移动 App 测试也面临着独特的挑战,它要确保 App 可满足其功能需求和非功能需求。^①

Vinson[Vin11] 建议,移动 App 的测试人员可适当调整测试 WebApp 所用的策略(第 25 章)。当然,测试内容时应考虑到移动设备和特定网络的局限性。由于设备特性和用户环境的多样化,兼容性测试和部署测试在移动领域中更加具有挑战性。性能测试需要确定在移动设备的有限存储、处理、连通性以及可用能力上,是否会对特性或功能性产生负面影响。移动 App 的性能测试经常是在某些细节层次上进行的,这只见于实时系统的开发。安全性测试必须考虑到物理设备的损耗,并且注意到一个事实,移动 App 经常是直接在移动设备硬件上运行,这就为窃取个人数据带来一定的显露度风险。移动 App 的设计往往考虑使用者是那些缺乏技术知识的人员,这与 Web 用户相比是不同的。这就需要进行更为广泛的用户体验测试(26.3 节)。敏捷开发过程模型(第 5 章)和测试驱动开发模型(第 24 章)均可采用。

关键点 为测试 Web 应用所采取的所有策略都是测试移动 App 的人员必须要采纳的。

570

26.2.2 对自动化的要求

移动 App 的测试人员常会遇到许多不同的配置(多种设备、多种操作系统和各种移动网络),还要保证移动系统的功能和运行环境是已知的。由于高效和全面地测试移动 App 十分重要,所以自动化地完成配置测试(25.7 节)或回归测试(22.3 节)就很有意义。不过还应该注意,要做到全面实现移动 App 的自动化测试(例如,在掌上视频游戏中的用户交互查找)是不可能的。

当测试人员不得不机械地处理大量重复性的测试用例时,自动测试工具便能提高团队成员的士气。越早使用自动测试工具,测试频度越高,就能越早地发现移动 App 的缺陷。敏捷开发过程(第 5 章)要求使用按日的构建周期,需用回归测试保证变更不会产生意想不到的负面影响。

移动通信公司(Mobile Labs, Inc.)提出了一种移动 App 的自动测试方法[Mob 11],内容包括:

可行性分析。若采用自动化测试,就应确知哪些测试和哪些测试用例具有最大的投资回报率(ROI),侧重点放在自动重复和经常使用的测试用例上,目标是争取做到手工测试用例的 50%~60% 实现自动化。

概念证明。确认测试自动化的价值。为做到这一点,要使有限数量的手工测试脚本自动化,以确定工作量的投资回报率(ROI)。测试团队必须确定其他脚本的可测量性以及在后继周期中的复用度。

测试框架的最佳实践。针对移动 App 提出了一种方法,以此作为其测试过程的基础。框架决定了移动 App 的实现规则和测试规则,框架是针对每个移动平台开发的,并且适合企业中的所有 App 的情况。

定制的测试工具。借助于先进的脚本技术,对每个移动平台(和被测的 App)定制测试工具。

引述 如今有几十个移动设备,但作为一名开发人员,为了和它打交道,你必须对每种设备能做什么有一定的理解。

John Fowler

提问 自动化移动测试最重要的是哪些方面?

① 对移动 App 测试特有问题的概述见 <http://www.utest.com/landing-interior/crowdsourcing-your-mobile-app-testing>。

571

在实际条件下测试。确知 App 如何在测试实验室之外的真实设备上运行。在真实设备上而不是在模拟器上进行测试能够降低假缺陷报告的发生率，并且更有可能发现用户级的错误（26.2.5 节）。

快速处理缺陷。通过自动提交缺陷信息和生成差异报告的加速实施，减少了开发人员处理缺陷的时间。

复用测试脚本。在考虑增强 App 的能力时，不必从头开始创建测试用例，复用可以帮助我们节省成本。要使功能界面和测试逻辑分离，测试工具的体系结构是很重要的。这就使界面包括在可复用的功能中，如同让工具作为新平台和新设备一样。

26.2.3 建立测试矩阵

移动 App 往往是为多种设备开发的，并且可用于不同的环境和不同的位置。加权设备平台矩阵（Weighted Device Platform Matrix, WDPM）有助于确保测试范围覆盖移动设备的各种组合和各种不同的环境^①。加权设备平台矩阵也可协助进行设备 / 环境组合的优先排序，以便于优先做最重要的测试。

为多个设备和操作系统建立加权设备平台矩阵（图 26-1）的步骤是：（1）矩阵的纵列给出各种重要的操作系统；（2）矩阵的横行是各种目标设备；（3）矩阵的第二行和第二列分配了排名次序（如从 0 到 10），这是为了表明每个操作系统和每种设备的相对重要性；（4）计算每对排名次序数的乘积，并填入矩阵（矩阵中有的排名次序对可能是无效组合）。

		操作系统 1	操作系统 2	操作系统 3
	排名	3	4	7
设备 1	7	无效组合	28	49
设备 2	3	9	无效组合	无效组合
设备 3	4	12	无效组合	无效组合
设备 4	9	无效组合	36	63

图 26-1 设备平台加权矩阵

要调整测试工作，使对于正在考虑的每个环境变量而言，具有最高评级的设备 / 平台组合得到最多的关注。在图 26-1 中设备 4 和操作系统 3 就具有最高评级，因此，在测试过程中将会获得最优先的关注。

26.2.4 压力测试

对移动 App 进行压力测试是要在极限运行条件下力图查找错误。此外，压力测试还提供了一种机制，在不损害安全性的情况下观察移动 App 的运行水平是否会降低。极限活动包括：（1）在同一设备平台上运行几个移动 App；（2）感染带病毒的或是恶意的系统软件；（3）尝试接管设备并使其传播垃圾邮件；（4）使移动 App 处理数量极大的事务；（5）在设备上存储异常大量的数据。遇到这些情况时，检查移动 App 以确保资源集中服务（例如流媒体）得以适当实施。

有效的压力测试 [Soa11] 应该“在自然环境下”进行，测试中会涉及

关键点

故障弱化是容错系统的重要特性。在出现错误时，如果关闭之前系统不能修复损害并允许继续执行，那么系统将使故障弱化，以期获得已知的安全状态。

① 环境变量是与当前连接或当前事务相关的变量，移动 App 将用来指导其可见的用户行为。

各种设备和各种操作系统,用两到三倍的额定量进行测试。测试应该在现实生活环境中反映用户的各种使用状况,包括本地和其他国家的不同网络配置和不同标准。

在以下几节中将更为详细地叙述这些原则。

26.2.5 生产环境中的测试

许多移动 App 的开发商主张进行自然环境测试,或是在用户的本地环境中使用移动 App 资源的生产发布版本进行测试。伴随着移动 App 的演变,自然环境测试是要敏捷地响应变更 [Ute12]。

自然环境测试的特征包括不利的和不可预测的环境、过时的浏览器和插件、独特的硬件以及不完善的联通性(无线网络和动态载流)。为了反映真实情况,测试人员的人口统计学特征应该与目标用户的特征及他们的设备特征相匹配。此外,应该包括涉及少量用户的用例、不太流行的浏览器以及各式各样的移动设备。自然环境测试总是会带有不可预测性,并且测试计划必须适应测试的进展。为了解更多的信息,可参看 Rooksby 和他的同事在他们论文中关于自然环境测试的成功策略 [Roo 09]。

引述 每个口袋
里都有手机这
种微型计算机,
这是重要的移动
设备。

Ben Horowitz

创建自用测试环境的过程是昂贵且易于出错的。基于云计算的测试可以提供标准化的基础架构和预配置的软件映像,使得移动 App 团队不必担心找不到服务器或是无处购买软件和测试工具的许可证。云服务提供商为测试人员的访问提供可扩展性,并为用户准备好虚拟实验室,其中有操作系统库、测试管理工具和执行管理工具,以及为生成能够准确反映现实生活的测试环境所需的存储器 [Myl 11]。

基于云的测试并不是没有潜在的问题,例如在使用云方法时,存在缺少标准、安全隐患问题、数据布局和完整性问题、不完备的基础架构支持、服务的不当应用以及性能问题等,但这也仅仅是开发团队所面临的一些共性挑战。

573

SafeHome 生产环境中的移动 App 测试

[场景] Doug Miller 办公室。

[人物] Doug Miller (SafeHome 软件工程技术团队经理); Vinod Raman (SafeHome 产品软件工程团队成员)

[对话]

Doug: 你认为我们的 SafeHome 移动 AppV0.0 版的电子商务部分怎么样?

Vinod: 外包商调整了 SafeHomeAssured.com 上的移动 App 来适应移动 App 环境,做得很好。Sharon (供应商的开发经理)告诉我,现在他们正在测试原型。

Doug: 我听说他们在用设备模拟器为电子商务网站做测试。我认为我们应该在实际设备上做些测试。

Vinod (扮了个鬼脸): 我以为应该请一个第三方测试公司来验证移动 App,好像我们还在对自己的产品赶尽杀绝!

Doug: 我们将请测试供应商来做性能测试、安全性测试以及配置测试。我们的外包商已经做了一些测试。我只是想换个角度,看来会是有益的,而且我们希望能控制成本,所以……

Vinod (叹了一口气): 你期望怎样呢?

Doug: 我想要确保用户有可靠的体验。

Vinod: 我认为可以从每一个主要的界面功能的用例开始。

Doug: 好的。只是跟踪它们的逻辑路径,从头一直到尾。检查加权设备平台矩阵。

我希望你能在顶层的六个最重要的设备上检查其性能，同时你还要检查出现在每个导航节点处的内容。当每个屏幕显示出内容时，确保设备的特性都考虑在内。

Vinod：当然要这么做。那么还有功能元素也要考虑。现在谁在测试可用性呢？

Doug：哦，测试厂商会调整可用性测试。我们已经聘请了一家市场调研公司，他们为可用性研究列出了20个典型用户。不过如果你发现了任何可用性问题……

Vinod：我知道，那就一起传给他们。

Doug（笑着说）：多谢了，**Vinod**。

26.3 与用户交互的各种情况

在功能相同的多种产品充满市场的情况下，用户自然会挑选易于使用的移动 App，其中用户界面及其交互机制是移动 App 用户的可见部分。移动 App 提供的用户体验质量测试能满足用户的期望，这是非常重要的。

在第 15 章中讨论的许多评估软件用户界面的可用性规程可以用来评估移动 App。同样许多用于评估移动 App 质量的策略（第 25 章）也可用来测试移动 App 的用户界面部分。人们还构建了更多良好的移动 App 的用户界面，这些并不是简单地从现有的个人计算机应用系统中缩放用户界面的尺寸而得到的。

提问 哪些移动 App 的可用性特征成为了测试的重点？这会涉及哪些特定目标？

信息栏 移动 App 可用性测试构件

在移动 App 测试博客中提供了许多有关移动 App 关键构件可用性测试的建议。^①

- **功能性**——确保用户故事支持核心功能性，确保利益相关者的目标和期望也在考虑之中。
- **信息架构**——确保内容和链接已结构化，并以逻辑形式呈现，而且数据块^②及其联系也在考虑之中。
- **内容**——使用文本、视频、图像和多媒体，只有当其在移动环境下支持用户任务时，才帮助用户控制是否启动媒体，以保证内容用移动设备的格式呈现。
- **设计**——设计屏幕的快速可视化扫描，同时考虑纵向和横向显示，重新考虑屏幕的布局，不只是缩放。
- **用户输入**——让用户易于输入数据，提

供自动完成和拼写检查，显示默认值，提供基于单个设备能力的交替输入机制。

- **移动环境**——说明环境的变更（一天中的某一时间、地点、网络）并利用设备特性和能力进行预测，并支持用户使用环境。
- **可用性**——确保交互设备（触摸屏、键盘、音频）和小部件（按键、链接、滚动条）都共同地向着目标设备协调运行，遵循约定并适应学习曲线。
- **可信性**——对保密性和安全性要敏感，未取得明确的用户许可之前，不收集个人信息，允许用户控制如何共享个人信息以及如何叙述商业活动。
- **反馈**——向用户提供重要信息，使报警

① 博客可见于 <http://www.mobileapptesting.com/10-key-components-of-successful-mobile-app-usability/2012/09/>。

② 数据块是指打破超媒体文件，使之成为更小的相关信息组，以利于读写器更快地进行吸收。

次数最小化，使报警信息简明，为用户操作提供确认而无需中断用户的工作流。

- 帮助——让用户易于访问帮助和支持选项，提供环境帮助。

26.3.1 手语测试

由于当前移动设备中非常流行触摸屏，因此，开发人员已添加了多种触摸手语（例如，刷新、缩放、滚动、选择等）作为扩展用户交互的可能性，这些手语不会造成屏幕损耗。然而不幸的是，手语密集界面带来了大量的测试挑战。

使用自动化工具测试触摸或手语界面操作是很难的。手语是难以准确记录以用于重现的。屏幕大小和分辨率以及之前的用户操作都会影响到屏幕上对象的位置。有时纸上原型会作为设计的一部分，但这并不能满足手语测试的要求。相反，测试人员需要开发测试框架程序，使其完成模拟手语测试的功能。但是这些做法都是既昂贵又费时的。

575

视障用户的可访问性测试是具有挑战性的，因为手语界面通常不提供任何触觉反馈和听觉反馈。对于智能手机之类的无处不在的设备，手语的可用性和可访问性测试是非常重要的。当手语操作无效时，测试设备的操作就更加重要。

用户故事或用例在理想上可以写得足够详细，使其可作为测试脚本的基础。补充有代表性的用户是非常重要的，包括所有目标设备在内都要进行补充，当使用移动 App 测试手语时，要考虑屏幕差异。最后，测试人员应确保手语符合为移动设备或平台设定的标准和环境。

26.3.2 语音输入和识别

目前智能移动设备可以使用语音输入，也允许同时进行设备的手忙操作和眼忙操作。语音输入可以采用几种形式，其中每个过程都带有所需的不同级别的编程复杂性。当消息记录只用于后期回放时，电子语音信箱输入便可以起作用了。离散字识别可以让用户从具有数量不多选项的菜单中，以口语提出选项。连续语音识别的目的是将口述语音直接转化为有意义的文本串。每一种口音的输入都将对其自身的测试构成挑战。

根据 Shneiderman[Shn09] 的理论，来自噪音环境的干扰会妨害各种形式语音的输入和处理。与指向屏幕对象或按键相比，使用语音命令来控制设备会给用户带来更大的认知负担。用户必须想出正确的字和词，以便移动 App 执行所需的动作。当屏幕上显示出一个对象时，用户只是要辨认出适合的屏幕对象，并将其选中即可。然而语音识别系统的广度和准确性还在迅速发展。在很多移动 App 中，语音识别很可能成为通信的主要形式。

建议 语音输入测试应该考虑到环境状况以及个人声音的变化。

即使对于最好的测试机构来说，测试语音输入和识别的质量和可靠性都是技术上的挑战。不正确的语音输入（由于用户错误而发出错音的字、词或是环境的干扰）都必须通过测试分辨出来，以确保不正确的输入不致造成移动 App 或设备的失灵。由于每个用户 / 设备组合都会有所不同，因此用户的广大人群和环境都应该考虑在内，以保证将差错率限制在可接受的范围之内。最后，记录错误也是重要的，这可以帮助开发人员提高移动 App 的语音输入能力。

576

26.3.3 虚拟键盘输入

由于激活虚拟键盘时可能会遮挡部分显示屏，因而应测试移动 App 以确保当用户键入时重要的屏幕信息不会被隐藏，这是十分重要的。如果必须隐藏屏幕信息以测试移动 App

的能力,则要让用户轻触页面,但并不丢失输入的信息。[Sch09]。

通常虚拟键盘比个人计算机的键盘小,因此很难用十个手指按键。由于键本身较小,难以准确敲击,而且并不提供触觉反馈,因此必须测试移动 App 以确保它易于纠错,并且在键入错误词语时不导致崩溃。

预测技术(即自动完成部分词语的输入)往往使用虚拟键盘来帮助用户加快输入。如果考虑要使移动 App 面向全球市场,针对用户选择的自然语言,测试输入词语完成后的正确性是十分重要的。同样重要的是测试任何机制的可用性,以允许用户可以不顾建议的做法。

通常虚拟键盘测试是在可用性实验室中进行的,但有些则应该在自然环境下进行。如果虚拟键盘测试发现了重要的问题,那么唯一的选择是确保移动 App 可以接受设备的输入,而不用虚拟键盘输入(如人工输入或语音输入)。

26.3.4 警报和异常条件

当移动 App 在实时环境中运行时,有许多因素在影响着它的行为。例如,当用户在使用移动 App 时,可能丢失无线网络信号,或是传入文本消息、电话呼叫,还可能接收的日历警报。

这些因素可能破坏移动 App 用户的工作流,然而大多数用户会允许弹出警报或是中断,因此,移动 App 测试环境必须能够模拟这些警报和条件。此外,在实际设备的工作环境中,应该测试移动 App 处理警报和条件的能力(26.5 节)。

移动 App 测试应该注重与警报和弹出消息相关的可用性。测试应该检查警报的清晰度和环境,检查这些事件在设备显示屏上出现位置的适当性,并且当涉及外语时,要验证一种语言翻译成另一种语言的正确性。

在各种移动设备上,由于网络或环境的变化,可能会引发许多不同的警报和条件,虽然许多异常处理过程可以用软件测试工具进行模拟,但在开发环境中,不能仅仅依靠模拟测试。这里再次强调实际设备在自然条件下测试移动 App 的重要性。

[577]

26.4 跨界测试

国际化是一个创建软件产品的过程,它使得在多个国家、操着各种语言来使用产品成为可能,而不需做任何工程的改变。本地化是调整软件应用以适应全球各地区使用情况的过程,通过添加各地的特定需求和把产品文本部件翻译为适用的语言来实现。除了语言差异以外,本地化还应考虑到不同国家的货币、不同的文化、税收和标准(包括技术标准和法律等)[Sla12]。因此,若要在世界许多地区投放和使用移动 App,则在这些方面进行测试显然是非常有必要的。

每个国家为实施本地化计划构建内部测试设施是非常昂贵的,相比之下,对每个国家本地供应商的外包测试则是较为划算的[Reu12]。然而,采用外包方式时,在移动 App 开发团队和实施本地化测试的供应商之间会有通信水平下降的风险。

众包(crowdsourcing)[⊖]在很多在线社区都很流行。Reuveni[Reu12]

引述 通过移动设备、社交、云计算、大数据、社区以及其他强大力量的融合,整个世界正在重塑。这些技术的结合开启了令人难以置信的机会,以一种全新的方式把一切都连接在一起,使我们的生活方式和工作方式发生着惊人的转变。

Marc Benioff

⊖ 众包是一种分布式问题解决模型,其中社区成员影响着指派给小组的问题解决方案。

建议在开发环境之外，众包可以供分布于全球的本地化测试人员使用。要做到这一点，发现声誉高且有成功业绩的社区是非常重要的。易于使用的实时平台使社区成员可与项目决策者沟通。为了保护知识产权，只有愿意签署保密协议且可信的社区成员才允许参加测试。

26.5 实时测试问题

由于在设备中实现的硬件和固件的组合，使得实际移动设备具有固有的局限性。如果潜在的设备平台范围很大，那么执行移动 App 测试将是昂贵的，而且还是耗时的。

设计移动设备时并没有考虑到测试。有限的处理能力和存储容量可能不允许装载所需的诊断软件以记录测试用例性能。而模拟设备则往往更易于管理，也更易于获取测试数据。

每个移动网络（全球有数百个）均采用自己独特的基本架构。移动网络运营商实现的网络代理指导用户如何连接到特定网络资源，以使用其网络。这可能会在服务器和测试客户端之间限制信息流传输。有些代理可能会在超文本传输协议（http）的头部信息中清除重要信息，其中包括你的 App 需要的功能或设备适配的信息。网络信号强度可能会成为其中的问题。模拟器经常无法模拟网络服务的效果和时序，当移动 App 在实际设备上运行时，可能观察不到用户发现的问题。

578

远程移动设备是一种有用的测试工具，使用它可以克服使用模拟器的一些局限性。远程移动设备是一个实际的移动手持机，安装在带有遥控器和遥控天线的盒子里。遥控器与设备屏幕和键盘控制电路相连。当连接到互联网时，这种组合方式允许本地个人计算机上的用户或是网络客户端的用户按下按钮，收集在远程设备上发生的数据。此外，为了随后可以重放，一些远程设备具有记录测试用例的能力，以此协助自动实施回归测试的过程。

最后，在移动设备上监测与移动 App 的使用特别相关的功耗是很重要的。与监测网络信号相比，从移动设备传输信息会消耗更多的电能。与加载网页或发送短信相比，处理流媒体会消耗更多的电能。因此，准确评估功耗必须在自然条件下的实际设备上实时监测。

26.6 测试工具和环境

在 26.3.2 节中讨论过，缩短测试时间、提高测试过程的质量和覆盖范围是将移动 App 测试的某些方面实现自动化的理由。同样，我们在 26.2.5 节中讨论了在生产环境中进行测试的重要性。不过，如果时间允许，手工测试还是需要的。但即使是在这种情况下，工具也可以用来监测跨网络设备上移动 App 和用户的行为。

当选择移动测试自动工具时，Khode[Kho12b] 建议采用以下几条准则。通常情况下这些准则也适用于移动测试工具。

- 对象识别。工具可以使用各种方法识别设备对象，这些方法可能是对象 ID、图像处理、文字识别、HTML5 或 DOM 对象。
- 安全性。工具不得要求使用未受保护的设备与公共互联网相连。
- 设备。工具利用实际用户的设备，无需使用专门的开发商模式。
- 功能性。支持所有设备的功能，包括多点触摸手语、虚拟键盘输入、呼入和手机短信、警报处理及其他。
- 模拟器和插件。在不同的设备和不同的移动操作系统上使用现有的测试环境，可执行同样的测试。

提问 我们应该采用哪些准则为移动测试选择自动化工具？

579

- 连通性。同时使用 USB、Wi-Fi、私有云和电话载波的多设备连接，以测试连接的稳定性和恢复性。

软件工具 为移动 App 测试选择工具

以下列出了可用于移动 App 测试的几种有用的工具。这是一个不断变化的领域。下面的工具较有代表性，是最近由 Brown[Bro 11] 和 Vinson[Vin 11] 推荐的。^①

移动网页工具要确定网页友好的移动设备的友好程度。用户输入网页的网址 (URL)，该工具会给出一个缺陷表。

- WC 3 mobile OKChecker, <http://validator.w3.org/mobile/>。

移动浏览模拟器在移动浏览器上显示网页的外观。用户会输入一个网页的地址 (URL)，工具会在移动浏览器上显示其外观。

- 手机模拟器, <http://www.mobilephonemulator.com/>。
- 苹果 iphoney, <http://www.maketcircle.com/iphoney/>。

设备模拟器通常是在个人计算机上运行的虚拟设备，可以用来开发和测试移动 App，而无需访问实体设备。

- 安卓模拟器, <http://developer.android.com/sdk/index.html>。
- ipad peek, <http://ipadpeek.com/>。
- Adobe Edge Inspect, <http://html.adobe.com/edge/inspect/>。
- BlackberrySimulators, <http://us.blackberry.com/sites/developers/resources/>

simulators.html。

自动化工具在 iOS 或 Android 上记录交互作用，作为允许重放的测试脚本。通常这些工具在带有设备模拟器的个人计算机上运行。

- MonkeyTalk, <http://www.gori-llallogic.com/testing-tools/monkeytalk>。
- Eggplant Mobile, <http://www.testplant.com/>。
- Device Anywhere, <http://www.Keynotedevicewhere.com/>。

网络监测工具添加、修改和过滤发送到网络服务器上的 HTTP 请求信息头。还可作为浏览器插件进行安装。

- 修改信息头, <http://addons.mozilla.org/en-us/firefox/addon/modify-headers/>。

移动分析测试用于收集数据，以分析用户如何与移动 App 进行交互，这对于评估投资回报率 (ROI) 是很重要的。通常需要网络或云服务以协助数据收集和存储。

- Flurry, <http://www.flurry.com/flurry-analytics.html>。
- Google 移动分析, <http://www.google.com/analytics/mobile/>。
- Distimo Monitor, <http://monitor.distimo.com/>。

580

26.7 小结

移动 App 测试的目标在于对多种移动 App 的质量进行检测，以找出错误或是发现可能导致质量事故的问题。测试会集中于若干质量元素，如内容、功能、结构、可用性、使用环

^① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在多数情况下，工具的名字由各自的开发者注册为商标。

境、导航性、性能、电能管理、兼容性、互操作性、容量和安全性。在完成移动 App 设计后，将进行评审和可用性评估，在应用实现后和在实际设备上部署时，就要进行测试。

通过对各“单元”内容、功能性或导航性的初步检查，移动 App 的测试可检测到每一维度的质量。在对每一部件进行验证以后，就将移动 App 作为一个整体进行测试。为做到这一点，许多测试是由用户的观点导出的，并且由用例中包含的信息驱动。移动 App 计划制定出来以后，就确定了测试步骤、工作产品（如测试用例）以及评估测试结果的机制。测试过程包含若干不同类型的测试。

内容测试（或评审）关注的是各类内容。其目的在于检查那些影响将内容展示给最终用户的错误。因移动设备限制而导致的性能问题也必须是检查的内容。界面测试检查交互机制和由移动 App 所提供的用户体验。这样做的目的是为了发现当移动 App 不考虑设备、用户或是位置等条件时所导致的错误。

导航测试基于用例，是作为建模活动的一部分而导出的。针对部署移动 App 所用的体系结构框架内的导航设计，以测试用例来运行每个场景。构件测试检查移动 App 中的内容和功能单元。

针对特定设备或网络环境，配置测试力图发现错误和兼容性问题。此后，测试是要发现与每个可能的配置相关的错误。由于移动设备和网络服务提供商的数量巨大，因此将使问题更复杂。安全性测试包含了许多针对移动 App 或其环境中薄弱环节的测试。安全性测试的目的是查找在设备操作环境中或是访问的网络服务中存在的安全性漏洞。性能测试是在需要服务器端的资源容量增加时，评估移动 App 的响应时间和可靠性。最后，移动 App 测试应该解决的性能问题包括：耗电量、处理速度、内在限制、从故障中的恢复能力以及连通性问题等。

581

习题与思考题

- 26.1 在某些情况下，是否可以忽略在实际设备上所作的移动 App 测试？
- 26.2 用自己的话说明测试移动 App 的目标。
- 26.3 兼容性是重要的质量因素。为了确保移动 App 的兼容性，必须要测试什么？
- 26.4 安装一款免费的移动 App 测试工具，对于你所熟悉的移动 App，评价该工具的性能。
- 26.5 移动 App 的什么元素可能在单元测试中得到测试？在移动 App 的元素集成以后才能进行什么类型的测试？
- 26.6 开发正式的书面测试计划是否总是必要的？请作解释。
- 26.7 全部移动 App 的测试策略都是以用户可见元素开始的，并且向技术元素发展。这种说法是否合理？这个策略是否有例外？
- 26.8 验证性测试（certification testing）在传统意义上真的是测试吗？请作解释。
- 26.9 描述对移动 App 所做的与用户体验测试有关的步骤。
- 26.10 假如你正在开发一个访问网上药店的移动 App，为的是满足老年人的购药需求。药店可提供典型功能，而且还为每位客户维护数据库，以便提供药物信息并且对可能的药物间潜在相互作用提出警告。针对这一移动 App 讨论特定的可用性测试。
- 26.11 假定您已实现了网络服务，为 YourCornerPharmacy.com 提供了药物互作用检查功能（参看习题 26.10）。探讨将要在移动设备上实施的构件级的测试类型，以保证移动 App 正常地使用这项功能。
- 26.12 如何考虑测试移动 App 的环境能力？

- 26.13 在生产环境中测试移动 App 时遇到的每一种配置都可能做到吗？如果不可能，应如何选择一组合理的测试配置？
- 26.14 描述 YourCornerPharmacy 药店（习题 26.10）的移动 App 可能要进行的安全性测试，谁应执行这种测试？
- 26.15 移动 App 的压力测试和 WebApp 的压力测试之间的差别是什么？

扩展阅读与信息资源

有许多描述移动计算的书籍，其中往往包含着对移动 App 测试的讨论。Kumer 和 Xie 的书（《Handbook of Mobile Systems Applications and Services》，Auerbach Publications, 2012）涵盖了移动服务以及移动计算中面向服务体系结构的作用。普及性计算读物定义了移动计算环境的原则，包括：Chalmers（《Sensing and Systems in Pervasive Computing: Engineering Context Aware Systems》，Springer, 2011）、Adelstein（《Fundamentals of Mobile and Pervasive Computing》，McGraw-Hill, 2004）以及 Hansmann（《Pervasive Computing: The Mobile World》，2nd ed, Springer, 2003）。在另外一书中，Nguyen 等人（《Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems》，2nd ed., Wiley, 2003）讨论了注重测试可理解性（test accessibility）、可靠性和安全性的移动 App 测试。

有关界面设计的书籍有很多，其中不少都包含有关测试移动 App 可用性方面的信息。Ben Shneiderman 和他的同事编写的书（《Designing the User Experience》，5th ed., Addison-Wesley, 2009）在可用性方面是一本最优秀的著作。其他较好的参考书有：Quesenberry 和 Szuc（《Global UX: Design and Research in a Connected World》，Morgan Kaufmann, 2011）以及 Schumacher（《Handbook of Global User Research》，Morgan Kaufmann, 2009）。Nielsen（《Mobile Usability》，New Riders, 2012）提供了有关如何设计可用界面的建议，其中也考虑到了移动设备的屏幕尺寸。Colborne（《Simple and Usable Web, Mobile, and Interaction Design》，New Riders, 2010）描述了简化用户交互的过程。Ginsburg（《Designing the iPhone User Experience: A User-centered Approach to Sketching and Prototyping iPhone Apps》，Addison-Wesley, 2012）讨论了采用以用户为中心的方法评估用户体验的重要性。

Meier（《Microsoft Application Architecture Guide》，2nd ed., Microsoft Press, 2009）提供了有关移动 App 测试的有用信息。而 Graham（《Experiences of Test Automation: Case studies of Software Test Automation》，Addison-Wesley, 2012）为测试自动化提供了良好的背景。Lee（《Test-Drive iOS Development》，Addison-Wesley, 2012）研究了在设计测试驱动的环境中移动 App 的测试过程。

在网上有各种各样关于移动 App 测试的信息源。最新的参考文献可在 SEPA 网站 www.mhhe.com/pressman 上的“software engineering resources”中找到。

安全性工程

要点浏览

概念：安全性工程师构建的是具有保护他们的资产免受攻击能力的系统。使用威胁分析可以确定在系统的弱点受到攻击时所需的控制措施，以减少显露度。安全性是软件质量要素的重要前提条件，比如完整性、可用性、可靠性和安全性都以安全为前提。

人员：软件工程师要和依赖系统成果或依赖于服务的客户以及其他利益相关者合作。

重要性：每一门新兴技术都可能引起用户对于隐私的关注，并且为其有价值的信息可能被窃取而担忧。安全性不只是软件开发人员的关注点，更是军事部门、政府机构以及卫生部门的关注焦点。如今每一位软件工程师都必须关注安全性，他们一定要注意保护好项目客户的资源。

步骤：首先识别系统资产，确定由于安全性漏洞造成的损失。在构件层上建立系统结构模型。然后，制定安全性需求规格说明和风险缓解计划。在系统建成后，实施安全性保证，并将其贯彻于软件过程的始终。

工作产品：主要的工作产品是安全性规格说明（可能是需求模型的一部分）和作为系统质量保证文档一部分的文档化安全性用例。为了开发这些工作产品还要建立威胁模型，并制定安全性评估计划以及风险缓解计划。

质量保证措施：使用安全性评审和安全检查所得结果以及测试结果作为安全性用例，使系统的利益相关者可评估系统在保护资产和维护私密方面的可信任度。

Devanbu 和 Stubblebine[Dev00] 对他们提出的安全性工程路线图曾做出如下说明：

是否有一种软件系统不再需要安全了呢？事实是：从互联网可知晓的在个人计算机上运行的客户端应用程序，到可通过互联网访问的复杂的电信系统和电力系统，再到具有版权保护机制的商用软件，几乎所有受软件控制的系统均会面临潜在对手的威胁。软件工程师必须意识到这种威胁，并且要设计出具有可靠防卫性的系统，同时还要为客户提交有价值的产品。

十年前就有人撰文讨论威胁问题，而后伴随着网络的爆炸式增长、无处不在的应用系统和云的广泛使用，这类论文更是蜂拥而至。所有这些技术都提出了关于用户隐私和个人信息可能丢失或是被窃取的新的忧虑。安全性问题不仅是软件开发人员的关注点，更是国防部门、政府和卫生机构关注的焦点。如今，每位软件工程师都必须关注安全性问题，以切实保护客户端的资源。

关键概念

资产
保证用例
攻击
模式
面
显露度
安全性
保证
用例
工程
模型
需求
威胁
分析
建模
信任验证

从最简单的道理来说,软件的安全性提供了使软件系统保护资产免于受到攻击的机制。对于任一利益相关者而言,资产是具有价值的资源。资产包括数据库信息、文件、程序、硬盘驱动器的存储空间、系统内存甚至处理器的容量。攻击通常是利用软件的弱点或漏洞,致使未授权人访问系统。例如,在允许用户访问有价值的系统资源之前没有发现的问题很可能会成为一个漏洞。

软件安全性是软件质量保证(第21章)的一个方面。从本书前面的章节中我们已经知道,质量是不能因响应隐错报告才加入系统中的。与此相似,安全也很难因响应已发现的系统漏洞而加入现有的系统中[Gho01]。安全性问题必须要在软件过程的开始就予以考虑,将其纳入软件设计之中,作为编码工作的一部分来实现,并且在测试和部署过程中加以验证。

本章中,我们提供了一份关于软件安全性工程中重要问题的调查。当然,对于这一主题的全面讨论已经超出了本书的范围。更多的信息可在[All08]、[Lip10]和[Sin08]中查阅。

引述 除非认识不足,否则安全性问题始终受到人们的极度重视。

Robbie Sin-clair

27.1 安全性需求分析

软件的安全性需求是由以下两方面确定的:一是与客户合作共同识别出的必须得到保护的资产;二是当出现安全性漏洞时,这些资产受损的成本。资产损失的价值被称为显露度。损失可用恢复或重建资产的时间和成本来度量。无关紧要的资产无需保护。

建议 一定要关注具有最高价值和最大风险显露度的资产。

构建安全系统的重要组成部分是可能被用于破坏系统资源的预计条件,或是可能被用于破坏系统资源的威胁,或是使授权用户无法访问的威胁。这一过程称为威胁分析。在系统资产、系统漏洞和威胁识别出以后,便可以制定控制措施,使系统既可避免受到攻击,又可缓解所遭受的破坏。

软件安全性是软件的完整性、可用性、可靠性和安全性(第19章)的必要前提,要想创建能防御资产免受所有可能威胁的系统是做不到的。因此必须鼓励用户维护好关键数据和冗余系统构件的备份副本,确保其安全保密。

585

SafeHome

利益相关者的安全性

[场景] 软件工程团队的工作区。

[人物] Jami Lazar、Vinod Raman、Ed Robbins, 软件团队成员; Doug Miller, 软件工程师; Lisa Perez, 营销团队成员兼产品工程代表。

[对话]

Vinod: 如果没有异议,由我来主持本次会议,可以吗?(在场各位都点头表示同意。)我们需要开始确定 SafeHome 项目的安全性问题。

Doug: 我们是否可以首先把大家所担心的那些防护事项列举出来?

Jamie: 好,如果一个外部黑客侵入 SafeHome 系统,企图抢劫或破坏主人的房子,会怎样?

Lisa: 如果有人知道我们的系统不能抵御黑客入侵,公司的声誉就将受损。

Jamie: 暂且不说责任,那样的事件发生了也会被认定是设计有缺陷。

Doug: 在传输密码时,产品的网站界面有可能使外人截获密码。

Ed: 更重要的是,网站界面需要包含客户信息的数据库,所以我们有对保密问题的担忧。

Vinod：也许这是个好时机，让每个人花十分钟的时间，列出各自认为受到攻击时可能会丢失或损坏的资产。

(10 分钟以后)

Vinod：好了，让我们把这些资产都贴在白板上，看看是否有类似的担忧。

(15 分钟后，白板上已经贴满了)

Lisa：看起来存在很多问题，那么我们怎

么处理好呢？

Doug：需要根据资产损失所造成破坏的修复成本，把列出的各项进行优先排序。

Lisa：怎么做呢？

Vinod：我们需要用到历史项目数据，获取替换损失资产的实际成本。Lisa 你要和法律部门联系，以得到我们可能要承担的赔偿责任。

27.2 网络世界中的安全性与保密性

互联网活动把传统的桌面浏览转移到场景中，其中浏览器提供了定制的动态内容。在合并了第三方网站数据的社区论坛上，用户可以输入自己的内容。很多桌面应用程序都利用网站浏览器界面访问本地数据，并在多种计算平台上提供相同的用户体验。因此，要求网站开发者提供更好的控制 and 安全性机制。不应为不可信源的数据和代码给予与可信程序员的代码相同的权限。为了使网站浏览器成为有效的用户界面，应该把保密、信任和安全当作最为重要的质量属性 [Sei11]。

在很多情况下，用户的机密信息流在因特网上都会跨越组织的边界。例如，患者的电子信息可能需要在医院、保险公司和医生之间共享。同样，旅游信息可能需要在旅行社、酒店和航空公司之间共享。在这些情况下，用户必须披露自己的个人信息以接受服务。当这些信息以数字形式被接收以后，用户通常无法控制组织用它做什么。在理想情况下，用户应当可以直接控制他们的数据信息以进行处理和共享，但是当这些数据电子化后，就需要由用户规定数据共享的优先权 [Pea11]。

27.2.1 社交媒体

在线社交媒体网络的迅速增长和普及，使它们成为吸引恶意程序人员的目标。由于默认被信任的大多数用户都处于社交网络环境中，因此黑客很容易利用受损的账户向账户持有人的朋友发送恶意的已感染信息。社交网络可能被用以引诱用户到钓鱼网站[⊖]，欺骗他们提交个人资料或是转发现金给急需的朋友。另一诡计则是利用含有详细信息的电子邮件窃取用户的个人信息 [Sae11]。

有些社交媒体网络允许用户开发自己的应用程序。这些应用程序只允许用户转让个人信息，而后以用户意想不到的方式使用这些信息。有些应用程序或是游戏足以吸引知识渊博的用户提供他们的个人信息，以便使用该应用程序。社交网络往往有签到功能，这就使得罪犯能够瞄准用户在现实生活中的活动作案。无论是身份盗取、垃圾邮件还是间谍软件问题，许多计算机用户都未能采取积极措施进行自我保护 [Sta10]。

引述 依靠政府来保护你的隐私，就像要求一个偷窥狂来帮你安装百叶窗一样。

John Perry
Barlow

586

引述 只要你把隐私披露出来，就不应该责备别人将其泄露出去。

Kahlil Gibran

⊖ 钓鱼网站伪装成知名且值得信任的网站，引诱用户提供个人信息，可导致安全资产的损失。

27.2.2 移动 App

移动 App 的用户可以使用与固定有线网络用户几乎相同的网络服务。无线互联网用户除继承了所有与桌面商务相关的安全风险以外，还外加了移动网络特有的新风险。无线网络要求节点之间的信任与合作。这可能会被恶意程序所利用而拒绝服务或收集机密信息。利用设备拥有者的所有权限，为移动设备开发的平台和语言都可能被黑客攻击，并且恶意代码也可能被插入设备的系统软件之中。这就意味着安全性技术（例如登录、身份验证和加密）都可能很容易地受到破坏。

提问 我们在移动 App 中会遇到什么威胁？

27.2.3 云计算

由于数据是委托给服务提供商管理的远程服务器的，因此云计算自然地带有更多的机密性和隐私问题。这些云服务的提供商拥有全面访问和控制我们信息的能力。相信他们不会与别人分享这些信息（无论是有意或是无意的），并且会采取负责的措施来防止发生损失。对于数据挖掘而言，在线数据存储库是非常诱人的信息源（例如，收集人口统计信息或市场信息等）。问题是数据始发者并没给出同意的信息 [Ray11a]。因此，策略制定者应该制定出政策和法规，以确保服务提供商不滥用用户的信任。

[587]

当一家公司采用云计算时，“内部置信”与“外部不置信”之间的边界会变得模糊。由于组织的应用程序和数据不再处于原位，因而可能出现一种新型的内部恶意人员。机密数据可能只用几条命令就被恶意者或是无资格的系统管理员非法访问。大多数云服务提供商在适当环境下均有严格的规定来监督员工访问客户数据。但针对远程攻击与监控时，防止数据遭受不法物理访问的策略并不高效，往往只是在事发之后才能检测到攻击。在云系统中，为取得用户的信任，为用户提供一些评估保护机密性和隐私的必要机制是否合适的手段是很重要的 [Roc11]。

无处不在的网络访问和云计算的出现为商业合作提供了全新的形式。但共享信息和机密的保护却是一项艰巨的任务。安全的多方计算提高了利己行为的风险，除非各方都有信心保证没有人会利用系统谋取私利。这种情况突出了人们对系统信任和安全性的心理空间的维度，不能只靠软件工程来解决 [Ker 11]。

27.2.4 物联网

富有想象力的人士是这样来描述“物联网”的 [Rom11]：物联网上的任何实在的东西在互联网上都有其虚拟的实体。这些虚拟实体可以提供服务，也可以消费服务，并且朝着一个共同的目标进行合作。例如，通过用户所用周边设备的网络，从一部手机上就可以了解到该用户的身体状况和精神状况；汽车工程师设想的轿车可以和其他车辆、数据源和设备之间进行自主通信，而无需驾驶员的直接控制。

然而，安全性是横亘在通往这个愿景道路上的一个主要障碍。如果没有强大的安全性基础，攻击和故障将会超过物联网的任何优势。策略的制定者必须考虑治理与创新之间的平衡。过度的治理可能很容易阻碍创新，反之，创新又可能在不经意间忽视人权 [Rom11]。

27.3 安全性工程分析

安全性分析包括需求获取、威胁建模、风险分析、测度设计和正确性检查。除去商业理由以外，这些任务包括系统的功能性和非功能性细节的考量 [Bre03]。

[588]

27.3.1 安全性需求获取

本书第 8 章讨论的需求获取的通用技术同样适用于安全性需求的获取。安全性需求是非功能性需求^①，它常常影响着软件系统的体系结构设计。使用威胁建模和风险分析将系统需求进行精炼和优化以后，就可以制定系统的安全策略了。除去考虑使用情况以外，为获取所需的安全性体系结构，将采用安全性建模和分解的方法，使这些策略得到细化。在这些策略实施前体系结构的安全性方面已得到确认 [Bod09]。

有些情况下安全性需求和其他需求是相互矛盾的。例如，安全性和可用性之间就有可能相互冲突。高度安全的系统会使那些没有太多经验的用户感到难以使用。在以用户为中心的安全性工程中，安全性需求获取对三个重要问题给出了回答 [Mar02]。这三个问题是：（1）对于安全性软件，用户的要求是什么？（2）如何设计安全体系结构，使其可提供良好的用户界面设计？（3）如何设计良好的用户界面，使软件不仅安全性好，同时还能使它运行起来有效、高效，并且让用户满意？应该把这些问题的答案与（第 8 章中讨论的）利益相关者和系统资源间的交互作用结合起来考虑。

在实施需求获取时，分析师应首先认清攻击模式。攻击模式是用于识别系统安全性缺陷的一种设计模式（第 16 章）。通过为常见的安全性漏洞提出问题和解决方案，攻击模式可加速安全性分析。复用攻击模式能帮助工程师识别系统漏洞。但没有必要重新设计不同的方式来攻击系统。针对软件安全性问题，攻击模式允许开发者使用易于理解的名称（例如，网络钓鱼、SQL 注入和跨网站脚本等）。常用的攻击模式可随时间的推移而得到改善 [Sin08]。当应用特定的模式时，使用攻击模式的困难之处是尽人皆知的。

一些软件工程师认为，在敏捷过程中安全性工程的严格性与需求获取的非正式特征是不相容的（第 5 章）。然而，有一项可用于调解“正式差距”的技术是在需求域内给出滥用者故事。滥用者故事基于客户输入来描述对系统资产的威胁。滥用者故事扩展了已建立的用户故事这一有效的敏捷概念，并且有助于实现安全性需求的可追踪性，使安全性保证得以持续进行 [Pee11]。

27.3.2 安全性建模

建模是说明需求和分析需求的一个重要过程。安全性模型是软件系统安全性策略的形式化描述。安全性策略提出了系统安全性的定义。安全性策略捕捉关键的系统安全性需求，同时将描述系统运行过程中如何加强安全性的规则包含在其中。

在设计、编码和评审过程中，安全性模型可以提供精确的指导。在系统建成之后，安全性模型提供了帮助验证安全性实施正确性的基础 [Dan09]。在维护活动中，安全性模型也是系统演化升级或维修的有价值的参考。

安全性模型可以用文字或图形来表示。无论安全性模型的表述形式如何，它都需要包含以下内容：（1）安全性策略的目标；（2）外部界面的需求；（3）软件安全性需求；（4）运行规则；（5）描述模型与系统对应关系的详细说明。

引述 用户在安全性问题上往往表现得很迟钝。

Bruce Schneier

提问 我们应该提出哪些关于安全性需求获取的问题？

引述 防卫计算机系统历来是一场智力的战斗，入侵者企图找到漏洞，而设计师们则设法封堵漏洞。

Gosser

589

① 有时称其为横切关注点，在第 4 章中曾进行了讨论。

有些安全性模型用状态机表示^①。每一状态都必须包括与系统安全性相关的信息，作为一名与安全性相关的软件工程师必须确保系统的任何状态都在安全状态下启动，并且在安全状态下结束。还必须验证初始系统状态是安全的。为了帮助人们完整地理解，模型需要有说明，以表明模型和实际系统的关系。

在验收之前，可执行的建模形式可让开发者验证安全性模型及其行为。在验收之后，模型便成为设计的良好基础。用于安全性需求建模的两种语言是 UML.sec（这是用构造型和约束对 UML 所做的扩展）和 GRL（用于捕捉非功能性需求的面向目标的需求语言）。在开发系统时形式化建模语言的使用有助于提高系统的可信度 [Sal11]。

作为系统的安全性分析和验证的扩展手段，人们提出了形式化方法（第 28 章）。在基于模型的安全性测试中，使用安全性需求的形式化规格说明可以帮助生成测试用例。对于关键的系统构件使用形式化证明可以增强开发者的信心，因为系统确实符合其规格说明。当然必须十分谨慎，要使证明的基本假设都得到满足。

590

27.3.3 测度设计

为了实现安全性，软件必须具有三种属性：可靠性（软件可以在不友好的环境下运行）、可信性（系统不会在恶意的方式下运行）和存活性（在已妥协的情况下系统可继续运行）^②。安全性度量^③和测度需要重点评估这些属性。

关键点 安全软件必须显示出三个属性：可靠性、可信性和存活性。

有用的安全性度量必须以测度为基础，使开发人员能够评估处于风险中的数据机密性和系统完整性可能达到的程度。生成此度量所需的一项测量是资产价值测度、威胁似然性测度和系统漏洞测度。这些属性都不易直接测量。损失资产的成本可能超过重建的成本。

最佳测度是在软件开发或运行期间现成的可用测度。办公桌上的安全投诉数量或安全性测试用例的失效数量可以提供一些测度（例如，每月报告的身份被盗事故的数量）。在攻击事件发生之前，我们可能并不知道有漏洞，但攻击得逞的数量是可以知道的。

27.3.4 正确性检查

安全性的正确性检查需要贯穿于整个软件开发周期。从对系统漏洞攻击的角度来分析，利益相关者资产的显露度应在开发过程的早期确定下来。

接着，软件团队要确保由系统用例导出的威胁模型已对风险缓解、风险监测和风险管理计划的安全性部分做出了解释。为在建模和构建活动期间使用，质量保证活动应包括安全性标准的制定和安全性指南的开发，软件验证活动应确保安全性测试用例是完备的，并可追溯到系统的安全性需求。

许多这类安全性检查均应包括在内植于常规软件工程任务的审核、审查和测试活动中（27.6 节）。正如在 27.4 节所述，对于在这些检查期间收

引述 理论上，我们可以建立可证明的安全系统。并且理论上也可以认为理论能够应用于实践。但实际上，这是不可能的。

M. Dacie

① 有限状态机是由一系列每一当前状态的可能转换状态和每个转换的触发条件定义的。

② <https://buildsecurityin.us-cert.gov>。

③ 度量是系统构件或过程具有特定属性的量化指标。良好的度量应满足 SMART（即特定的、可测量的、可得到的、可复用的和依赖时间的）标准。度量通常是利用统计技术来展示关系，通过实施测量而得到。有关度量测量的进一步讨论见本书第 30 章。

集到的数据进行分析，并且概括为系统安全性用例的一部分。可信性验证过程见 27.7 节的讨论。

591

27.4 安全性保证

如今软件已融入了我们的日常生活，安全上的缺陷和与之相关的损失变得更为昂贵，同时也更为危险。完善的软件工程实践包含明确的需求和相应设计的开发，从而表明确已开发出适用的产品。安全性保证是为了向最终用户和其他利益相关者表明确已开发出一个安全产品，从而增强他们的信心。

27.4.1 安全性保证过程

验证是保证任务的一部分，它提供证据以表明利益相关者可以确信这款软件是符合需求的。在安全性工程的环境下考虑问题时，选择安全性需求的关键子集或软件的索赔要求，并提出保证案例，以证明该软件是能满足这些需求和索赔要求的。

保证案例是已经讨论过和审查过的材料，它所支持的论点是，软件可满足正在维权的索赔。保证案例曾长期用于软件的安全性 (safety)，目前则正在用于软件的安全性 (security)^①。于是常常称作安全性用例。

关键点 安全性用例支持索赔要求，因此说明这样的软件是安全可靠的。

每个安全性用例包含三个要素：(1) 索赔要求本身；(2) 通过证据和假设使多个索赔要求之间彼此相连的论点；(3) 支持论点的证据主体和明确的假设。

为使安全性用例是有效的，三个目标必须得到满足：必须说明索赔要求对于该系统来说是适当的和可负担的；适用于工程实践的文件已得到实践，因而索赔要求是可以完成的；表明索赔的成果在风险要求的等级之内 [Red10]。

几种证据类型可用来证明安全性用例。如果用证明其正确性的意图来设计代码，那么软件正确性的形式化证明 (第 28 章) 或许是有帮助的。有些工具支持自动软件验证 [DSi08]，而另一些工具则实施软件安全性漏洞的静态扫描 (例如，RATS、ITS4、SLAM)^②。但是工具本身不能构建安全性用例。

一些证据可从对系统制品做类似于正式技术评审或审查得到 (第 20 章)。然而，这些评审只关注安全性的索赔要求。有安全性知识专长的人员可以评审系统或是安全性用例。检查单评估也可用来验证安全性指南和过程是否得到实施。

592

SafeHome 构建安全性用例

[场景] 软件工程团队的工作区。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Bridget Thornton，软件质量组组长。

[对话]

Ed：Bridget，感谢您来参加我们的讨论，我们正要为 SafeHome 项目构建安全性用例。

① 本章所涉及的 security 与 safety 的差别在于，前者包含保守秘密的意思，因此有人称其为“安密性”或“保密性”。——译者注

② 安全性工具的清单请参见 <http://www.tech-fag.com/how-to-find-security-vulnerabilities-in-source-code.html>。

Vinod：我们从哪里开始讨论呢？

Bridget：我们可以挑选一个存在安全性隐患的部件，看看我们能够找到什么证据来支持这个说法。

Ed：是什么证据呢？

Bridget：还是先挑选存在安全性隐患的部件吧！

Vinod：让我们集中在与客户数据库有关的安全性薄弱点上。

Bridget：好，让我们从所列出的访问数据库的索赔要求开始吧！

Jamie：你的意思是安全性模型的有些成分与数据库有关联？

Bridget：是的，接下来我们看看已完成的审查工作，还有就是正式技术评审的摘要，这个摘要应该是在完成这个项目里程碑时就做好的。

Ed：那么过程审核和小组提交的变更请求

文件又怎么考虑呢？

Bridget：这些都很重要，最好也考虑在内。

Vinod：由独立测试组（ITG）生成并且运行多数系统测试用例。

Bridget：归纳安全性测试用例的特点，比较其预期的输出和实际的输出，就可以得到安全性用例的非常重要的组成部分。

Jamie：这里面可能有大量的信息需要处理。

Bridget：是的，这正是下一节我们为什么为了数据库安全性提出各项索赔要求，并且总结出证据来支持或反驳有足够资产保护的索赔要求。

Ed：在汇总材料时，您能够帮助我们评审安全性用例吗？

Bridget：当然可以。这个项目启动以后无论是进度超前还是滞后，我们小组都要和你们团队进行持续沟通。

27.4.2 组织和管理

由于急着要把软件推向市场，使得项目经理往往更为关注项目的特性和功能，而把安全性列入次要地位。软件工程师应当重视软件设计和体系结构的健壮性。但除此之外，在构建基于软件的系统时，应当采用更为安全的做法 [Sob10]。

安全性保证和安全性风险识别活动要做计划、管理和跟踪，这和其他软件工程活动是一样的。软件团队要收集数据（例如，非法访问数、系统失效数、数据记录丢失数等），以确定什么活动起作用，什么活动不起作用。这就要求开发人员分析所报告的每一个失效（以确定失效的原因是否与系统漏洞相关），然后评估所导致的资产显露度。

27.5 安全性风险分析^①

识别和管理安全性风险是重要的项目计划任务（第31章）。安全性工程是由软件团队和其他利益相关者识别出的风险所驱动的。风险影响着项目管理和安全性保证活动。

威胁建模是一种安全性分析方法，可用于识别那些最有可能引发基于软件系统的破坏的威胁。威胁建模是在项目的初期阶段利用需求和分析模型完成的。建立威胁模型的工作包括：识别应用的关键构件、分解应用、构件威胁的识别和分类、对每个构件威胁的分级与分类、根据风险大小的排序将构件进行分级以及制定缓解风险的策略。微软采用以下步骤生成威胁模型 [Sin08]：

关键点 威胁建模是一种安全性的分析方法，可用于识别那些最有可能引发破坏的威胁。

^① 对软件项目风险分析的一般性讨论包括危及项目的所有类型的风险。其成功的分析方法在第35章中给出。

构建威胁模型需要什么步骤？

1. **确认资产。**列出所有的敏感信息和知识产权、存储位置、存储方式以及谁有访问权。
2. **给出体系结构概述。**写出系统用例并建立系统构件模型。
3. **分解应用。**目标是保证在应用构件之间发送的所有数据都是有效的。
4. **确认威胁。**使用如攻击树或攻击模式等方法，记录可能危及系统资产的所有威胁，其过程往往包括寻找网络、主系统配置和应用威胁等。
5. **记录威胁。**制作一个风险信息表，详细列出要监测和缓解的每项威胁。
6. **评估威胁。**对于处理可能的威胁，多数项目所提供的资源都显得不足，因此要根据其影响大小和发生的可能性做出排序，以便区别对待。

贵重的资产应该得到更好的保护，以防高概率风险的破坏。量化风险评估过程（第35章）可用于风险排序。首先需确认所有要评估的资产，并且要确定损失或恢复重建的资金价值。对每一项资产列出主要的威胁表，并用历史数据来确定在一个典型年份里可能发生的每项威胁。针对每项资产计算出每年每项主要威胁可能损失的金额以及每年损失的预期值（Annual Loss Expectancy, ALE）。最后，将与每个单独的威胁相关的ALE值相加，计算出损失每项资产的综合威胁。

594

SafeHome 安全性工作步骤

[场景] 软件质量保证组工作区。

[人物] Jamie Lazar、Vinod Raman，软件团队成员；Bridget Thornton，软件质量组组长。

[对话]

Vinod：嗨！Bridget 希望我们讨论安全性风险分析。

Bridget：讨论这个有助于建立开发工作的安全性优先级吗？

Jamie：我认为是的。

Vinod：我们是否可以着眼于数据库的安全问题？

Jamie：是的，通过历史数据，我们得知成本是备份数据和维护数据的记录。但是大家却不知道，如果客户的数据被盗，我们可能不知道应该给予的责任损害赔偿是多少，尽管对于这些成本我们有行业数据。

Jamie：那就是我们要求的吗？

Bridget：是的，你已经有了系统体系结构图。要验证在已被确认的构件之间交换的所有数据是比较容易的。我们还必须确定

每项资产的威胁。

Vinod：那么该怎么做呢？

Bridget：我们可以创建一个攻击树，从在根部设置攻击目标开始。例如，攻击者的目标可以是窃取客户信息。

Vinod：还有呢？

Bridget：然后查看数据库的攻击模式目录，找出适合的可作为攻击树的子目标。

Jamie：然后怎么做？

Bridget：你得要细化威胁，然后为每个威胁列出风险信息表，描述威胁的影响，任何监控措施和缓解措施都应到位，这样才能使问题得到解决。

Vinod：那么如何做才能有助于建立开发的优先性顺序呢？

Bridget：利用历史数据，对每一项威胁计算每年损失的预期值（ALE），便可确定每项威胁的成本，这部分过程我们可以提供帮助。

Jamie：多谢，Bridget。我们在识别和精选威胁后，作ALE计算时会寻求您的输入信息。

27.6 传统软件工程活动的作用

在发现系统运行低效且经常出现故障之后,就需要考虑构建新系统,使其安全运行。然而,有些软件开发人员认为,直到威胁明显暴露以前,系统中的威胁是无法预测的。因此,直到测试阶段,他们总是忽视安全性问题。这样做只是靠一时填补漏洞,以清除在软件过程早期阶段形成的安全性失误。以特定的方式对已有系统添加安全性补丁,而不对系统的设计或体系结构做大的变更,这是不可能的。因此,添加补丁的办法既是低效的也是昂贵的。

在开展任何开发工作之前,迭代过程和增量过程的性质(第4章)使其难以解决所有的安全性问题。而且软件需求常常在开发过程中出现变更。此外,体系结构设计的决策可能对安全性的关注点产生直接影响。正因如此,在项目开始时,很难处理好所有的安全性问题。即使大部分安全问题预先已得到解决,软件过程后期的设计决策仍然会影响最终系统的安全性漏洞 [Mei06]。

595

有效的软件过程包括一组合理的评审和调整措施。许多安全性活动都有互补作用,并且对软件质量具有协同效应。例如,众所周知,在测试之前,代码评审可以减少产品缺陷的数量,同时还可以消除潜在的安全性漏洞,从而提高软件的质量。

在制定计划时,项目预算和时间安排必须把安全性问题考虑在内,使得满足系统安全性目标^①的所需资源得到适当安排。作为安全性和保密性风险评估的一部分,每一项功能需求的要求都要检查,以便了解其是否会影响与系统安全性目标相关联的资产。在风险分析时,应确定和估计与每项损失相关联的成本。

确定处理系统特定威胁的机制常常延迟到将软件增量需求转化为其设计需求之后。这是因为在此情况下应该先确定攻击面。所谓攻击面(attack surface)是指存在于软件产品中的一组可获取并可利用的漏洞。许多安全性漏洞的交汇点将会被发现。例如,当其跨网络传播到数据库服务器时,在用户界面上以某种形式输入的信息可能被拦截。这样便可开发包括直接涉及攻击面的安全性规定在内的设计指南。

这可有助于区别安全性评审和一般的设计评审。侧重于安全性问题的代码评审应作为实现活动的一部分。应当根据系统设计活动中确定的相应安全性目标和威胁进行代码评审。

安全性测试是系统测试(第22章)的常规部分。安全性风险评估可以作为测试用例的来源,使得安全性测试更加受到关注。应急响应计划(Incident Response Plan, IRP)阐明了每个系统利益相关者要开展的活动,以响应特定的攻击 [Pra07]。应急响应计划的充分评审应该是安全性验证过程的一个组成部分。

此外,验证应包括安全性操作和资产归档规程的评审。安全性风险管理计划应作为维护过程的一部分进行定期的评审。

在应用软件部署以后,报告出现安全性事件时,作为系统维护的一部分,开发人员应评

引述 世界上人类知识的总量大概每十年翻一番,我们的安全性知识只能靠学习能力的提高来得到。

Nathaniel
Branden

关键点 攻击面被定义为软件产品中一组可获取的和可利用的漏洞。

提问 什么是应急响应计划?

引述 当你认为自己能够处理所面临的任何事务时,那是因为你拥有世界为你必须提供的安全性。

Harry Browne

① 例如,客户数据的保护,与系统信息的机密性、完整性和可用性相关的法律和法规要求的认可,以及其他知识产权的保护等。

估安全性风险管理规程的有效性(第 36 章)。如果系统变更规程(第 29 章)包括根本原因分析,这可能有助于发现在整个系统设计中的漏洞。

596

27.7 可信性系统验证

当我们在软件安全性的语境中考虑问题时,信任表明一个系统实体(或是一个组织)对另一系统实体(或组织)相信的程度。系统实体包含整个系统、子系统和软件构件。信任具有心理维度和技术维度。一般来说,假设第二个实体的表现正如第一个实体所期望的那样,第一个实体可以说信任第二个实体。论证这一假设的正确性是验证系统可信性的任务。虽然已经提出了多个信任模型 [Sin08],但我们仍然关注确保系统在威胁模型中符合所提出的缓解风险的实际方法。

关键点 “信任”

表明一个系统实体(或组织)对另一实体(或组织)相信的程度。

验证工作能确保使用基于测试、审查和分析技术的特定和可量化度量,以评估可信性系统的需求 [She10]。测试度量包括检测到的故障次数与预测的故障次数之比,或已通过的安全性测试用例数与运行的总数之比。其他度量还包括正式评审活动的缺陷排除效率(第 32 章)。在分析活动中确保安全性测试用例回到已开发的安全性用例的可追溯性也是很有益的。

对于信任实体的所有协作者来说,用来证明安全性用例的证据必须是可以接受的和有说服力的。可信系统的用户应该确信系统没有可被利用的漏洞或恶意的逻辑。作为验证任务的结果,当遭到损害时,用户应在系统的可靠性和可存活性上充满信心。这便意味着对软件的损坏已降到最低限度,并且系统能够很快恢复到可接受的运行能力。具体的安全性测试用例和规程也是验证过程的一个重要组成部分 [Mea10]。

SafeHome 安全性测试用例的生成

[场景] Vinod 的工作区。

[人物] Vinod Raman 和 Ed Robbins 都是软件团队成员。

[对话]

Vinod: 为了异地访问 SafeHome 系统的视频,我们需要提出安全性测试用例。

Ed: 我们应该从评审 Doug 和 Bridget (软件质量组组长) 开发的安全性用例开始。

Vinod: 我认为可以让独立测试组 (ITG) 承包商做这项工作,但这似乎是非常简单的测试用例。同时为了回归测试,应当把它加到我们使用的测试用例组中。

Ed: 好的。用例密码要求用户登录到一个网站,使用一个有效的 ID 和两个密码,并且在请求视频按要求反馈后,用户要输入一个四位数字识别码。

Vinod: 这给了我们几个逻辑路径进行测试,

有四个用户输入的数据。每个输入都需用一个正常的值、一个不正确的值、一个空值和一个格式不正确的值进行测试。

Ed: 为覆盖所有的逻辑路径需要用 256 个不同的测试用例。

Vinod: 是的,的确是这样。我们还需要对每个用例的响应进行定义。

Ed: 基于安全性策略,针对每一条信息,用户可以尝试三次。

Vinod: 对,而且在每一次尝试失败后,提示用户输入数据。

Ed: 并且要在任何一条信息的第三次尝试中也失败,则系统应该发送电子邮件给公司和用户,以使他们警觉。

Vinod: 把测试用例以随机排序的方式提交给密码检查员可能是个好办法。我们可能要要不只一次地运行我们的测试用例,确

597

信密码检查人员对历史记录并不敏感。

Ed: 我们应该写一个小程序, 运行所有这些测试用例并记录结果。

Vinod: 是的, 这其中有大量的工作。也许

我们应该让独立测试组 (ITG) 与 Bridget 的软件质量保证组一起来开展安全性测试。

如今, 软件质量测量已无法充分满足信任保证和安全性的要求。现有的测度 (例如, 考虑失效之间平均时间的可靠性 (reliability) 测度或是测量缺陷密度可信性 (dependability) 测度) 往往都忽略了很多因素, 这就可能使得软件受到损害, 并且容易受到攻击。在某种程度上这是因为许多度量并没有考虑到一个现实问题, 这就是存在一些活跃的不良人员在不断地挖掘软件漏洞。

在涉及信任的情况时, 基于实体过去的行为, 有效的安全性度量能够保持历史数据。例如, 当一个电子商务网站让它的买家和卖家作评价时, 就会考虑到所建立的信任。当然, 这类评级体系必须确保对被评价的实体有恰当且合理的确认, 而且有关实体并没有不准确的记录数据。这些问题有时会困扰信任报告系统。

美国国土安全部提倡采用安全软件设计的做法, 采用可靠而标准化的测量工具。理想的情况下, 这些开发工具的使用可以帮助开发者减少引入系统的漏洞数量 [Mea10]。这可能使那些被信任系统的用户在导引下做出有关系统可信性的决策。但正如系统可靠性那样, 用户也可能根据使用系统时所受到损失的程度做出判断。

软件工具 安全性工程

[目标] 安全性工程工具帮助人们在源代码中找到安全性漏洞。

[机制] 为了方便开发人员仔细检查, 通常让工具阅读源代码并标记编程结构来处理软件源代码。

[代表性工具]^①

- RATS (Rough Auditing Tool for Security) 由 Secure Software 开发 (<http://code.google.com/plrough-auditing-tool-for-security/>)。这是一个扫描工具, 它为安全性分析师提供了一组潜在的故障点, 以及问题的描述和建议的补救措施。
- ITS4 由 Cigital 开发 (<http://freecode.com/>

[projects/its4/](http://projects.its4/))。这是一个为寻找漏洞而对关键的安全性 C 源代码和 C++ 源代码作静态扫描的工具。

- SLAM 由 Microsoft 公司开发 (<http://research.microsoft.com/en-us/projects/slam/>)。该工具检查软件是否满足其使用界面的关键行为属性, 并且在设计界面和软件时帮助软件工程师确保实现可靠且运行安全。
- 许多安全性源代码的扫描工具可参看 <http://www.tech-fag.com/how-to-find-security-vulnerabilities-in-source-code.html>。

27.8 小结

软件安全性工程涉及资产保护, 以使软件开发活动免受威胁的损害。威胁可能涉及攻

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在多数情况下, 工具的名字由各自的开发者注册为商标。

击,即利用系统漏洞危害系统服务或数据的机密性、完整性或可用性。

安全性风险管理涉及评估可能的威胁影响,获取安全性需求,使关键的损失降到最小。安全性设计包括开发系统体系结构以使引入的已知漏洞数量最少。作为减轻损失影响一种手段,软件工程师要利用技术来防止攻击、击退攻击并且从攻击中恢复系统。

为鼓舞利益相关者的信心,需要开发人员把安全性保证作为普适性活动,在整个软件过程一开始就要予以考虑。但安全性度量的开发仅处于初期阶段。为系统构建的安全性用例包括收集使用安全性测试的证据,以及开展以安全性为重点的正式技术评审,目标是确保安全性指南和风险缓解措施已得到和执行遵循。

习题与思考题

- 27.1 考虑你自己的手机 App。首先简要地描述一个 App,然后列出至少 3~5 个安全性风险。
- 27.2 针对上一题中提到的一个风险,描述一个安全性缓解策略。
- 27.3 列出经常可能用于攻击 WebApp 的 5 种攻击模式。
- 27.4 描述应用于易趣(eBay)等投标网站上的信任模型。
- 27.5 为基于云的照片库描述安全性需求。
- 27.6 同源策略中哪些与可信赖系统有关?
- 27.7 对于个体消费者来说,利用互联网确定单独发生身份被窃事件的年度平均成本。
- 27.8 在系统完成后,若试图解决安全性风险,请解释可能遇到的一些问题。
- 27.9 利用互联网找出用于生成网络钓鱼攻击模式的详细信息。
- 27.10 为在数据服务器上丢失的数据计算年度损失的预期值(ALE),其重建价值为 30 000 美元。因黑客的攻击每年数据损失为 5%,而潜在损失达到 20 000 美元。

599

扩展阅读与信息资源

以下书籍都涉及重要的安全性问题: Vacca (《Computer and Information Security Handbook》, 2nd ed. Morgan Kaufman, 2013)、Goodrich 和 Tamassia (《Introduction to-Computer Security》, Addison-Wesley, 2010)、Anderson (《Security Engineering》, 2nd ed., 2008)、Kern 等人 (《Foundations of Security》, Apress, 2007)、McGraw (《Software Security》, Addison-Wesley, 2006) 以及 Dowd 和他的同事 (《The Art of Software Assessment: Identifying and Preventing Software Vulnerabilities》, Addison-wesley, 2006)。Zalewski (《The Tangled Web: A Guide to Securing Modern Web Applications》, No Starch Press, 2011) 以及 Howard 和 LeBlanc (《Writing Secure Code》, 2nd ed., Microsoft Press, 2003) 的书都讨论了如何构建安全系统。Allen 等人 (《Software Security Engineering》, Addison-wesley, 2008) 提供了项目经理的观点、Howard 和 Lipner (《The Security Development Lifecycle》, Microsoft Press, 2006) 讨论了安全性工程过程以及 Schumacher 等人 (《Security Patterns》, Wiley, 2006) 对使用模式作为安全性工程的一个有效要素提出了他们的见解。

其他一些书籍侧重于系统“入侵”方面,例如: Barnett 和 Grossan (《Web Application Defender's Cookbook: Battling Hackers Protecting Users》, Wiley, 2012)、Ottenheimer 和 Wallace (《Securing the Virtual Environment: How to Defend Against Enterprise Attack》, Wiley, 2012)、Studdard 和 Pinto (《The Web Application Hacker's Handbook》Wiley, 2011)、Howard 和他的同事 (《24 Deadly Sins of Software Security》, WcGrau-Hill, 2009)、Erickson (《Hacking: The Art of Exploration》, No Starch Press, 2008)、Andrews 和 Whittaker (《How to Break Web Software》, Addison-wesley, 2006) 以及 Whittaker (《How to

Break Software Security》, Addison-Wesley, 2003)。这些书都为软件工程师提供了有益的启示,因为他们也需要从系统和应用软件是如何遭受攻击这个方面对安全性问题加深理解。

此外, Sikorski 和 Honig (《 Practical Malware Analysis 》, No Starch Press, 2012) 以及 Ligh 等人 (《 Malware Analyst's Cookbook 》, Wiley, 2010) 对不良软件的内部事务给出了出色的洞察。Swiderski (《 Threat Modeling 》, Microsoft Press, 2004) 对威胁建模做了详尽的讨论。

有些作者在书中给出了他们实施安全性测试的指导原则, 这些书包括: Allen (《 Advanced Penetration Testing for Highly Secured Environments 》, Packet Publishing, 2012)、O'Gorman (《 Metasploit: The Penetration Tester's Guide 》, No Starch Press, 2011)、Faircloth (《 Penetration Tester's Open Source Tool Kit 》, Syngress, 2011)、Engebretson (《 The Basics of Hacking and Penetration Testing 》, Syngress, 2011)、Faircloth (《 Penetration Tester's Open Source Tool Kit 》, Syngress, 2011)、Hope 和 Walther (《 Web Security Testing Cookbook 》, O'Reilly Media, 2008) 以及 Wysopal 等人 (《 The Art of Software Security Testing 》, Addison-Wesley, 2006)。

网上有各种安全性工程的信息可供利用。基于模式设计相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

形式化建模与验证

要点浏览

概念：你曾听人说过多少次“第一次就将事情做对”？如果我们在软件开发中做到了这一点，就可以大大降低不必要的返工成本。两种高级的软件工程方法——净室软件工程方法和形式化方法——可以帮助软件工程团队“第一次就将事情做对”，这两种方法提供了基于数学的建模方法，并具有验证模式正确性的能力。净室软件工程强调在程序构建开始之前进行正确性的数学验证，并且将软件可靠性认证作为测试活动的一部分。形式化方法运用集合论和逻辑符号体系描述事实（需求）的清晰陈述，通过对这种数学规格说明的分析，可以提高（甚至证明）正确性和一致性。两种方法的底线是创建具有极低故障率的软件。

人员：特殊训练的软件工程师。

重要性：错误导致返工。返工需要时间，并增加成本。如果我们能够大量地减少在软件设计和构建中引入的错误（bug）

的数量，不是很好吗？这正是形式化建模和验证的前提。

步骤：使用数学验证易于处理的特殊符号创建需求和设计模型。净室软件工程使用盒结构表示方法，盒结构在特定的抽象级别上封装系统（或系统的某些方面）。一旦盒结构设计完成，即开始正确性验证。对每个盒结构完成了正确性验证后，则开始进行统计使用测试。形式化方法将软件需求翻译成更形式化的表示，方法是使用符号和启发式规则为系统功能定义数据不变式、状态及操作。

工作产品：开发特殊的、形式化的需求模型。记录形式化的正确性证明和统计使用测试的结果。

质量保证措施：形式化的正确性证明应用于需求模型。统计使用测试方法应用于测试使用场景，以保证发现和改正用户功能方面的错误。

一旦完成了软件模型和代码的开发，就要进行评审和测试。形式化建模和验证与评审和测试有所不同，形式化建模和验证是融合到特殊的建模方法中的，这些建模方法往往与规定的验证方法集成在一起。没有合适的建模方法，就不可能完成验证。

本章和附录 3 讨论两种形式化建模与验证方法——净室软件工程和形式化方法。两种方法都要求特殊的规格说明方法，并且每种方法都适合于一种独特的验证方法。两种方法都非常严格，都没有被软件工程团体广泛使用。但是，如果要构建出非常健壮（子弹打不穿）的软件，那么这些方法给你的帮助会是巨大的，因此值得学习。

关键概念

盒结构规格说明
认证
净室设计
净室过程模型
正确性验证
设计细化
设计验证
形式化方法
功能规格说明
统计使用测试

净室软件工程（cleanroom software engineering）是一种在软件开发过程中强调在软件中建立正确性要求的方法。与传统的分析、设计、编码、测试和调试的周期观点有所不同，净室方法的观点 [Lin94] 如下：

净室软件工程背后的哲学是：通过在第一次正确地书写代码增量，并在测试前验证它们的正确性来避免成本很高的缺陷消除过程。它的过程模型是在代码增量积聚到系统的同时进行代码增量的统计质量验证。

通过强调证明正确性的要求，净室方法在很多方面将软件工程提升到另一个层次。

在形式化方法（formal method）中，使用说明系统功能和行为的形式化语法和语义来描述所开发的模型。规格说明是以数学形式表达的（例如，谓词演算可作为形式化规格说明语言的基础）。Anthony Hall [Hal90] 在介绍形式化方法时给出了下面的论述（同样适用于净室方法）：

形式化方法（和净室软件工程）是有争议的。支持者声称该方法可以引发软件开发的革命；而批评者认为这是极端困难的。同时，对大多数人来说，他们对形式化方法（和净室软件工程）很不熟悉，因此难于判断这些争论。

在本章，我们概要介绍形式化建模和验证的概念。附录 3 将探索形式化建模和验证方法的某些技术细节。

28.1 净室策略

净室方法使用第 4 章所介绍的增量过程模型的专业版。一个“软件增量的流水线” [Lin94] 由若干独立的小型软件团队开发。每当一个软件增量通过认证，它就被集成到整个系统中。因此，系统的功能随时间增加。

每个增量的净室任务序列在图 28-1 中给出。在净室增量的流水线中，需要完成以下任务：

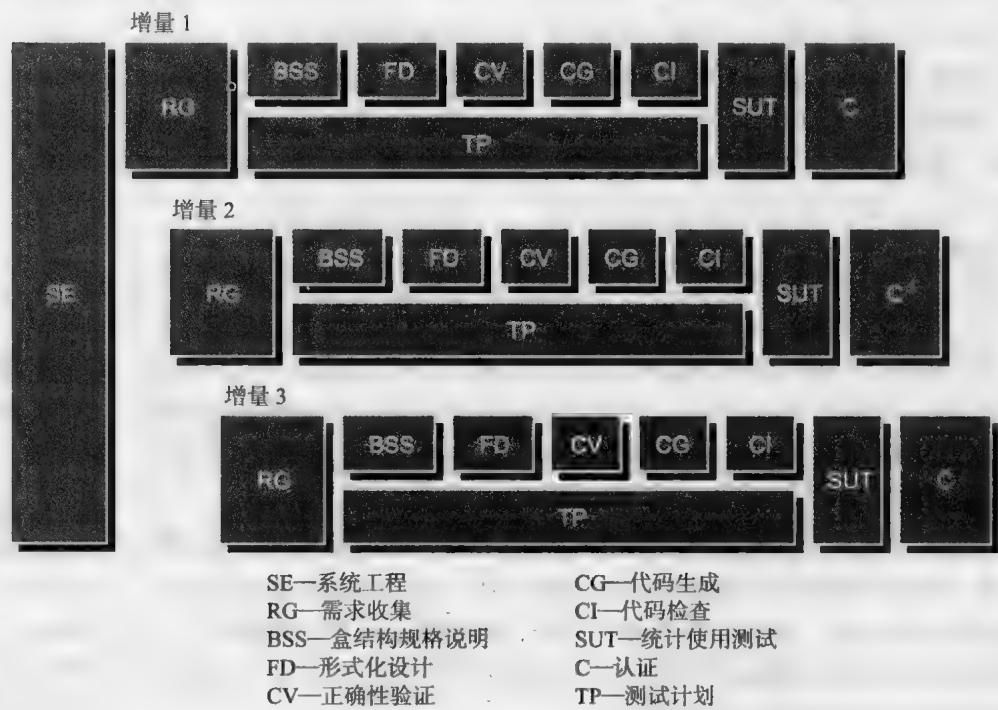


图 28-1 净室过程模型

- **增量计划。**制定一个采用增量策略的项目计划，确定每个增量的功能、预计规模及净室开发进度。必须特别小心，以保证通过认证的增量能够及时集成。
- **需求收集。**使用类似于第8章介绍的技术，（为每个增量）开发更详细的客户级需求描述。
- **盒结构规格说明。**运用盒结构的规格说明方法描述功能规格说明。盒结构“在每个细化级别上使用行为、数据及规程的构造性定义独立”[Hev93]。
- **形式化设计。**使用盒结构方法，净室设计是规格说明的自然无缝扩展。虽然清楚地区分两个活动是可能的，不过对规格说明（称为黑盒）进行迭代细化（在一个增量内）类似于体系结构设计和构件级设计（分别称为状态盒和清晰盒）。
- **正确性验证。**净室团队对设计及代码进行一系列严格的正确性验证活动。验证（28.3.2节）从最高层次的盒结构（规格说明）开始，然后移向设计细节和代码。正确性验证的第一层次通过应用一组“正确性问题”[Lin88]来进行，如果这些不能证明规格说明是正确的，则使用更形式化的（数学的）验证方法。
- **代码生成、检查和验证。**将某种专门语言表示的盒结构规格说明翻译为适当的程序设计语言。然后使用技术评审（第20章）来保证代码和盒结构的语义相符性，以及代码的语法正确性。最后，对源代码进行正确性验证。
- **统计测试计划。**分析软件的预计使用情况，计划并设计一组测试用例，以测试使用情况的“概率分布”（28.4节）。如图28-1所示，这个净室活动是和规格说明、验证及代码生成并行进行的。
- **统计使用测试。**回想一下，对计算机软件进行穷举测试是不可能的（第23章），因此，有必要设计有限数量的测试用例。统计使用技术[Poo88]执行由统计样本（上面提到的概率分布）导出的一系列测试，这里的统计样本是从来自目标人群的所有用户对程序的所有可能执行（28.4节）中抽取的。
- **认证。**一旦完成验证、检查和使用测试（并且改正了所有错误），则对增量进行集成前的认证工作。

净室过程的前4项活动确定了后面的形式化验证活动阶段。因此，下面开始讨论具有建模活动的净室方法，这些建模活动对于将要应用的形式化验证非常重要。

28.2 功能规格说明

在净室软件工程中，建模方法使用盒结构规格说明方法。一个“盒”在某个细节层次上封装了系统（或系统的某些方面）。通过逐步细化的过程，盒被细化为层次，其中每个盒具有引用透明性，即“每个盒规格说明的信息内容足以定义其细化信息，不需要依赖任何其他盒的实现”[Lin94]。这使得分析员能够按层次划分系统——从顶层的基本表示到底层实现的特定细节。净室软件工程使用三种类型的盒：

提问 净室项目使用哪种过程模型？

引述 程序中出现错误的唯一方式是作者将错误引入其中。没有其他方式……正确实践的目标是：设法避免引入错误，如果引入了错误，通过测试或任何其他运行程序的方式来消除错误。

Harlan Mills

602

603

建议 净室方法强调按软件被真正使用的方式进行测试。用例为测试计划过程提供了输入。

关键点 “盒”在某种抽象级别上对系统进行封装，并用于功能规格说明。

- **黑盒**。黑盒刻画系统行为或部分系统的行为。通过运用由触发映射到反应的一组转换规则，系统（或部件）对特定的触发（事件）做出反应。
- **状态盒**。状态盒以类似于对象的方式封装状态数据和服务（操作）。在这种规格说明视图中，表示出状态盒的输入（触发）和输出（反应）。状态盒也表示黑盒的“触发历史”，即封装在状态盒中且必须在蕴含的转换间保留的数据。
- **清晰盒**。在清晰盒中定义状态盒所蕴含的转换功能，简单地说，清晰盒包含了对状态盒的过程设计。

图 28-2 给出了使用盒结构规格说明的细化方法。黑盒 (BB_1) 定义了对完整触发集合的反应。可以将 BB_1 细化成一组黑盒，即 $BB_{1,1}$ 到 $BB_{1,n}$ ，每一个黑盒关系到一类行为。细化过程一直继续下去，直到可以标识出行为的内聚类（例如 $BB_{1,1,1}$ ）。然后对黑盒 ($BB_{1,1,1}$) 定义状态盒 ($SB_{1,1,1}$)。在这种情况下， $SB_{1,1,1}$ 包含了实现 $BB_{1,1,1}$ 定义的行为所需的所有数据和服务。最后， $SB_{1,1,1}$ 被细化为清晰盒 ($CB_{1,1,1,n}$)，并且刻画出过程的设计细节。

引述 对每个问题的每个解决方案都是简单的。神秘之处就在于两者之间的距离。
Derek Landy

当进行每一步细化时，也同时进行正确性验证。需要对状态盒规格说明进行验证，以保证每个规格说明均同其父黑盒规格说明定义的行为相一致。类似地，对照父状态盒，对清晰盒规格说明进行验证。

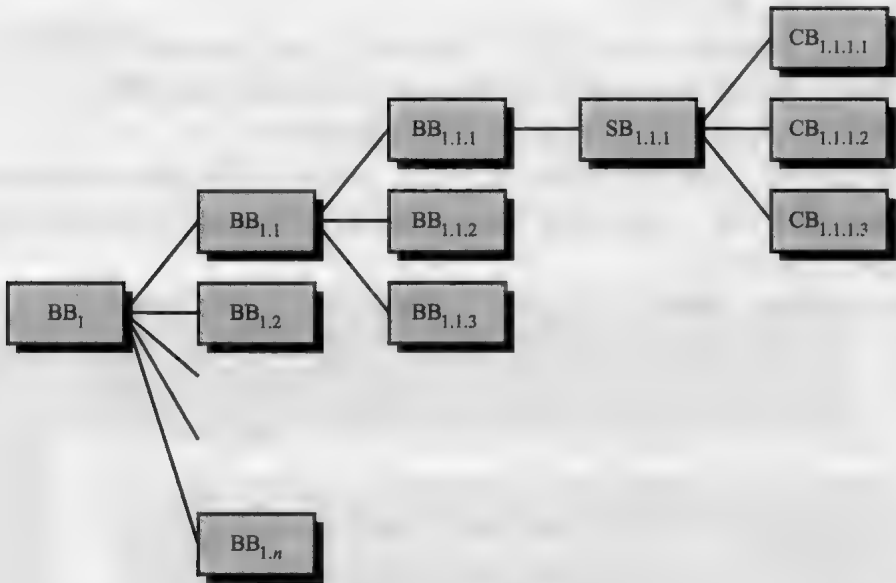


图 28-2 盒结构细化

28.2.1 黑盒规格说明

使用图 28-3 所示的符号 [Mil88]，黑盒规格说明描述一种抽象、触发和反应。函数 f 被应用到输入（触发） S 的序列 S^* ，并将它们变换为输出（反应） R 。对于简单的软件构件， f 可以是一个数学函数，但一般情况下会使用自然语言（或形式化规格说明语言）描述 f 。

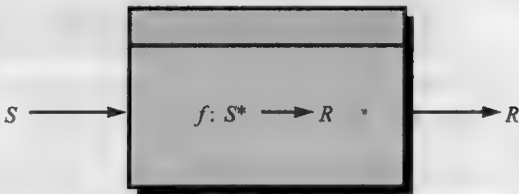


图 28-3 黑盒规格说明

为面向对象系统引入的很多概念也适用于黑盒。黑盒封装了数据抽象和操纵抽象数据的操作。和类层次一样，黑盒规格说明可以展示使用的层次，其中，低层盒从树结构中的高层盒继承属性。

28.2.2 状态盒规格说明

状态盒是“状态机的一种简单泛化”[Mil88]。回想第 11 章对行为建模和状态图的讨论，状态是某个可观察到的系统行为的模型。在进行处理时，系统对事件（触发）做出反应，从当前状态转换到某一新的状态。当进行转换时，可能发生某个动作。状态盒使用数据抽象来确定到下一个状态的转换以及状态转换后将要发生的动作（反应）。

如图 28-4 所示，状态盒可以与黑盒 g 结合使用。来自某外部源及一组内部系统状态 T 的触发 S 被输入到黑盒中。Mills[Mil88] 给出了包含在状态盒内的黑盒函数 f 的数学描述：

$$g : S^* \times T^* \rightarrow R \times T$$

这里 g 是和特定状态 t 连接的子函数。整体考虑时，状态-子函数对 (t, g) 定义了黑盒函数 f 。

606

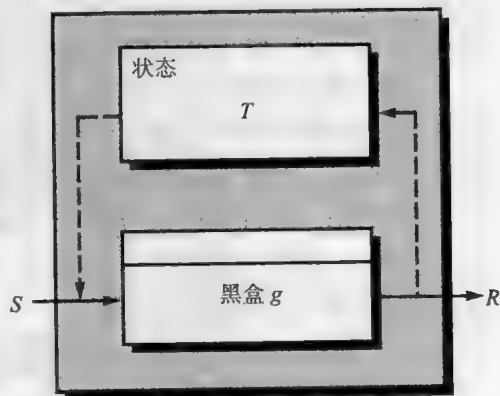


图 28-4 状态盒规格说明

28.2.3 清晰盒规格说明

清晰盒规格说明是与过程设计及结构化编程紧密关联的。实质上，状态盒中的子函数 g 被实现 g 的结构化编程结构所替代。

例如，考虑图 28-5 所示的清晰盒。在图 28-4 中的黑盒 g 被带有条件的顺序结构所替代。随着逐步细化的进行，这些结构又可以依次被细化为更低层次的清晰盒。

值得注意的是：在清晰盒层次中所描述的过程性规格说明可证明是正确的，这一主题在 28.3 节讨论。

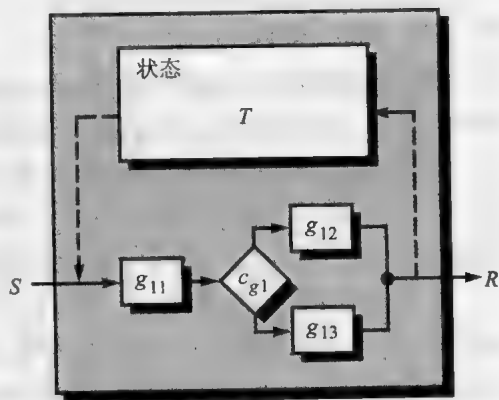


图 28-5 清晰盒规格说明

28.3 净室设计

净室软件工程主要运用结构化程序设计原理（第 14 章）。但是，在这里结构化程序设计应用得更严格。

对基本的处理函数（在规格说明的早期细化中描述）进行细化，其方法是“将数学函数逐步扩展为逻辑连接词（如 if-then-else）和子函数构成的结构，这种扩展一直进行下去，直到所有标识出来的子函数可以用程序设计语言直接表达实现”[Dye92]。

使用结构化程序设计方法对函数进行细化很有效，但是，对数据设计如何呢？这里，可以使用一组基本的设计概念（第 12 章），程序数据被封装为由子函数提供服务的一组抽象。使用数据封装、信息隐藏和数据类型概念进行数据设计。

607

28.3.1 设计细化

每个清晰盒规格说明代表了一个完成状态盒转换所需的过程（子函数）的设计。在清晰盒中，使用结构化程序设计结构和逐步细化表示过程细节。例如，一个程序函数 f 被细化为子函数 g 和 h 的序列，这两个子函数又进一步细化为条件结构（例如，if-then-else 和 do-while）。进一步细化直到具有足够的过程细节来创建所需要的构件。

在每个细化层次，净室团队^①执行一次形式化正确性验证。为此，将一组通用正确性条件附加到结构化程序设计结构上。如果函数 f 被扩展为序列 g 和 h ，则 f 所有输入的正确性条件是：

- 执行 g 之后再执行 h 能完成 f 的功能吗？

如果一个函数 p 被细化为形如 if $\langle c \rangle$ then q else r 的条件形式，则对 p 的所有输入的正确性条件是：

- 只要条件 $\langle c \rangle$ 为真， q 能完成 p 的功能吗？只要条件 $\langle c \rangle$ 为假， r 能完成 p 的功能吗？

如果一个函数 m 被细化为循环 do n while $\langle c \rangle$ ，则对 m 的所有输入的正确性条件是：

- 能够保证循环终止吗？
- 只要 $\langle c \rangle$ 为真，循环执行 n 之后能完成 m 的功能吗？只要 $\langle c \rangle$ 为假，退出循环仍能完成 m 的功能吗？

每当一个清晰盒被细化为下一个详细层次时，都应用上面给出的正确性条件。

关键点 如果在进行过程设计时，只使用结构化程序设计结构，就可以用一组简单的问题来证明代码的正确性。

28.3.2 设计验证

值得注意的是：结构化程序设计结构的使用限制了必须进行的正确性测试的数量。对顺序结构只检查单个条件，对 if-then-else 结构只测试两个条件，对循环则只验证三个条件。

为了举例说明过程设计的正确性验证，我们使用由 Linger、Mills 和 Witt[Lin79] 给出的一个简单的例子。目的是设计并验证一个小程序，该程序对某给定的整数 x ，找出其平方根的整数部分 y 。过程设计用图 28-6 给出的流程图来表示^②。

为了验证该设计的正确性，我们必须定义如图 28-6 所示的入口和出口条件。入口条件说明 x 必须大于或等于 0；出口条件要求 x 保持不变， y 满足图中描述的表达式。为了证明设计的正确性，需要证明图 28-6 表示的条件 init、loop、cont、yes 和 exit 在所有情形下都是正确的。有时将这些证明称为子证明（subproof）。

1. 条件 init 要求 $[x \geq 0 \text{ and } y = 0]$ 。基于问题的需求，假定入口条件是正确的^③。因此，init 条件的第一部分 $x \geq 0$ 是满足的。在流程图中，在 init 条件前的语句设置 $y = 0$ ，因此，init 条件的第二部分也是满足的，因此，init 为真。
2. 条件 loop 可能以两种方式之一出现：（1）直接从 init（此时直接满足 loop 条件）或（2）通过穿过条件 cont 的控制流。因为条件 cont 与条件 loop 相同，因此，不管从哪条路径到达它，条件 loop 都为真。

关键点 为了证明设计正确性，你必须首先标识所有条件，然后证明每个条件取合适的布尔值。将这些证明称为子证明。

① 由于整个团队都参与验证过程，因此实施验证本身产生错误的可能性很小。

② 图 28-6 来自 [Lin94]，已得到使用许可。

③ 在这里，负数的平方根没有意义。

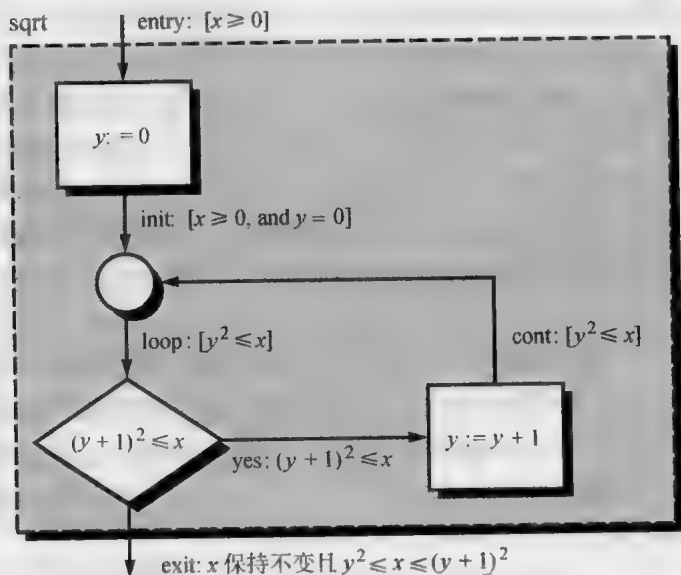


图 28-6 计算平方根的整数部分 [Lin79]

3. 只有在 y 值被递增 1 后, 才能遇到条件 **cont**。另外, 只有在条件 **yes** 也为真时, 才能调用到达条件 **cont** 的控制流路径。因此, 如果 $(y+1)^2 \leq x$, 则 $y^2 \leq x$, 条件 **cont** 成立。

609

4. 在如图所示的条件逻辑中测试条件 **yes**, 因此, 当控制流沿着所示的路径移动时, 条件 **yes** 一定为真。

5. 条件 **exit** 首先要求 x 保持不变, 对设计进行检查可发现 x 没有出现在赋值操作的左边, 没有使用 x 的函数调用, 因此, x 保持不变。因为条件测试 $(y+1)^2 \leq x$ 不成立时, 才可能到达条件 **exit**, 因此 $(y+1)^2 > x$ 成立。此外, **loop** 条件还必须保持为真 (即 $y^2 \leq x$), 因此, $(y+1)^2 > x$ 和 $y^2 \leq x$ 可以组合在一起满足 **exit** 条件。

我们必须进一步保证循环终止。对循环条件的检查显示: 因为 y 是递增的, 而 $x \geq 0$, 因此, 最后循环一定终止。

上面的 5 步是图 28-6 中算法设计的正确性证明, 我们现在可以肯定, 这个设计将计算出平方根的整数部分。

针对设计验证的更严格的数学方法是可能的, 然而, 这个话题的讨论超出了本书范围, 有兴趣的读者可参考 [Lin79]。

28.4 净室测试

净室测试的策略在根本上不同于传统测试方法 (第 22 ~ 26 章)。传统测试方法导出一组测试用例, 以发现设计和编码错误; 净室测试的目的是通过证明用例 (第 8 章) 的统计样本的成功运行来确认软件需求。

引述 质量不是
条例, 而是习惯
的成果。

Aristotle

28.4.1 统计使用测试

计算机程序的用户没有必要了解设计的技术细节。程序用户可见的行为通常是由用户产生的输入和事件所驱动的。但是, 在复杂系统中, 输入和事件的范围 (即用例) 是非常广泛

的。什么样的用例子集能够充分验证程序的行为？这是统计使用测试关注的第一个问题。

统计使用测试“等同于以用户试图使用软件的方式来测试软件”[Lin94]。为了完成测试工作，净室测试团队（也称为认证团队）必须确定软件的使用概率分布。对软件的每个增量的规格说明（黑盒）进行分析，并定义一组使软件改变其行为的触发（输入或事件）。通过与潜在用户的交流、使用场景的建立以及对应用领域的全面了解，为每个触发分配一个使用概率。

610

按照使用概率分布为每个触发集合生成测试用例[⊖]。为了举例说明，考虑本书前面讨论过的 SafeHome 系统。在此系统中，使用净室软件工程方法开发软件增量来管理用户与安全系统键盘的交互。对这个增量，已经标识出 5 个触发。通过分析，给出每个触发的概率分布百分数。为了更容易选择测试用例，这些概率被映射为 1 ~ 99 的数字区间 [Lin94]，如下表所示。

程序触发	概率 (%)	区间	程序触发	概率 (%)	区间
Arm / disarm (AD)	50	1 ~ 49	Test (T)	15	79 ~ 94
Zone set (ZS)	15	50 ~ 63	Panic alarm	5	95 ~ 99
Query (Q)	15	64 ~ 78			

为了生成符合使用概率分布的使用测试用例序列，生成 1 ~ 99 之间的随机数。每个随机数与前面概率分布的一个区间相对应。因此，使用测试用例序列可以随机定义，但又与触发生的适当概率相对应。例如，假定生成了下面的随机数序列：

13-94-22-24-45-56

81-19-31-69-45-9

38-21-52-84-86-4

根据表中显示的分布区间选择适当的触发，从而导出下面的用例：

AD-T-AD-AD-AD-ZS

T-AD-AD-AD-Q-AD-AD

AD-AD-ZS-T-T-AD

测试团队执行这些测试用例，并对照系统的规格说明来验证软件的行为。记录测试所用的时间使得我们可以确定间隔时间。利用间隔时间，认证团队可以计算出平均失效时间 (MTTF)。如果进行一个很长的测试序列后没有出现故障，则 MTTF 较低，软件可靠性较高。

611

28.4.2 认证

本章前面讨论的验证和测试技术可用来对软件构件（和完整的增量）进行认证。在净室软件工程方法中，认证意味着可以描述每个构件的可靠性（用 MTTF 来度量）。

可认证的软件构件的潜在影响远远超出了单个净室项目的范围，可复用的软件构件可以和它们的使用场景、程序触发以及概率分布一起存储。在描述的使用场景和测试体系下，每个构件都具有一个经过认证的可靠性。对于那些希望使用这些构件的人来说，这些信息是十分宝贵的。

建议 即使你决定使用净室方法，将统计使用测试作为测试策略不可分割的一部分也是值得考虑的。

关键点 不同于传统的测试，净室方法是统计驱动的。

⊖ 可使用自动化工具完成此项工作。进一步的信息请参见 [Dye92]。

认证方法包括 5 个步骤 [Woh94]: (1) 必须创建使用场景; (2) 指定使用概貌; (3) 从概貌中生成测试用例; (4) 执行测试, 记录并分析失效数据; (5) 计算并认证可靠性。步骤 1 ~ 步骤 4 已在前一节中讨论过。净室软件工程的认证需要创建三个模型 [Poo93]:

- **取样模型。**软件测试执行 m 个随机测试用例, 如果没有错误发生或只有少于指定数量的错误发生, 则通过认证。用数学方法导出 m 值, 以保证能够获得需要的可靠性。
- **构件模型。**对由 n 个构件组成的系统进行认证, 构件模型使得分析员能够确定构件 i 在完成前失效的概率。
- **认证模型。**设计并认证系统的整体可靠性。

在统计使用测试完成后, 认证团队已得到了交付软件所需要的信息, 包括使用上面的每个模型计算出来的经过认证的 MTTF。有兴趣的读者参阅文献 [Cur86]、[Mus87] 和 [Poo93] 可获得更多的详细信息。

28.5 重新思考形式化方法

大多数软件工程师都会有同感: 按照建模、设计、编码和测试范型开发没有缺陷的软件系统, 即便能够做到, 也是非常困难的。有些系统在部署之前不能对其进行充分测试 (例如, 在敌对环境下遥远星球上的机器人操作)。

形式化方法提供了验证关键系统按照其规格说明运作的方法。为了做到这一点, 我们将软件看作数学实体, 其正确性可以用逻辑操作来证明。事实表明: 与用面向对象语言实现的事件驱动的应用系统相比, 对于用命令式程序设计语言^①实现的程序, 其正确性的证明更容易。

使用形式化方法开发没有缺陷的系统有很多潜在的优势 [Abr09]。用自然语言书写的需求通常具有歧异性或是不完整的。形式化方法将系统建模为一系列的状态转换, 将程序运行时系统开发者可观察的东西表示出来。系统建模行为会揭示出系统中的多方面缺陷。大型软件应用系统的建模需要多次迭代。

水平方向的细化通过增加细节使软件状态从抽象变为具体。这种水平方向的细化是获得软件需求可追踪性的基础。可以定义断言以防止软件进入无效状态, 并对其进行验证。一旦完成了离散模型水平方向的细化, 就可以使用垂直方向的细化对状态和过程进行变换, 使得我们可以用目标编程语言来实现软件。垂直方向的细化是试图将抽象和具体“粘合”到一起的过程。不允许使用选择不当的目标语言, 这类语言存在的交流缺陷会影响需求规格说明。

遗留代码的整合使得形式化方法的使用复杂化了, 因为遗留需求可能不适合新系统了。在很多情况下, 最好是在规格说明中抓取遗留代码的行为, 并在新的代码中实现。

形式化方法的一些反对者认为很多形式化方法的实践与敏捷过程模型的原理相矛盾。然而, Black 和他的同事 [Bla09] 认为可以将形式化方法和敏捷过程的元素进行合并, 以此创建更好的软件产品。两者都具有同样的基本目标, 那就是创建可靠的软件。形式化方法通过强迫开发者保证系统安全特性公理^②的有效性来增加敏捷开发的值。

形式化方法技术 (例如, 静态分析和定理证明工具) 可用于从系统模型中自动产生测试

提问 我们如何对软件构件进行认证?

[612]

引述 形式化方法在被不理解它的人使用之前不会有重要的影响。
Tom Melham

① 命令式程序设计语言通过给代数表达式赋值而起到作用。

② 安全特性公理是关于不允许软件做什么的陈述。在第 27 章中将安全作为安全防卫的一部分进行讨论。

用例，并指明应该将断言放在进化程序代码的什么位置。可以将非形式化需求翻译成形式化的符号，并将其嵌入到源代码中。形式化符号书写的断言可以由机器检查其不一致性。随着代码的进化，经常需要对其进行重构^①。形式化方法为保持正确转换的定义提供了基础，以确保重构后的代码仍然满足需求。

尽管形式化方法的使用可能会延缓第一个软件增量的移交，但在项目生命期内能够降低返工的工作量，因此，对于所投入的时间能够提供可观的回报。如果客户的需求非常易变，那么使用敏捷技术也不能保证项目更快地完成。博弈的名义是创建满足客户需求的产品，并用形式化方法对敏捷方法进行补充，这种做法有助于确保需求得到满足。

Meyer 和他的同事 [Mey09] 描述了三种类型的工具，可以用于自动化测试，从而便于运用统计使用测试方法。测试生成工具产生并运行测试用例，而不需要人为输入。测试提取工具生成测试用例，用于程序运行失败时的重放。手动测试集成工具有助于开发和管理手工生成的测试用例。

测试中最有可能自动化的部分是测试用例的执行，这对有效的回归测试非常重要。Meyer[Mey09] 认为：对于某些语言来说，测试用例的生成比较直接，如支持契约式设计^②的 Eiffel 语言。可以产生测试用例来实施作为源代码一部分的前置条件和后置条件（在 28.6 节讨论）。当开发者发现了程序中的 bug 时，通常会进行记录和修复，但丧失了创建测试用例进行回归测试的机会。当程序运行中出现错误时，测试工具抓取有关前置条件不满足或后置条件失败的信息，并自动将其转换为测试用例。

软件工具 形式化方法

[目标] 形式化方法工具的目标是辅助软件团队进行规格说明和正确性验证。

[机制] 工具的机制各异。一般情况下，工具辅助进行规格说明和自动正确性证明，一般通过定义特殊的语言进行定理证明。很多工具没有商业化，而是为了科研目的而开发的。

[代表性工具]^③

- ACL2 由得克萨斯大学 (www.cs.utexas.edu/users/moore/acl2/) 开发，“它既是

可以用于对计算机系统建模的程序设计语言，同时又是帮助证明这些模型特性的工具”。

- 几个形式化方法工具库的链接可以在网站 <https://sw.theclsiac.com/databases/url/key/53/57#.UHRvKYbwpuh/> 找到，此网站由网络安全与信息系统信息分析中心 (Cyber Security and Information Systems Information Analysis Center, CSIAIC) 托管。

28.6 形式化方法的概念

《The Encyclopedia of Software Engineering》[Mar01] 中对形式化方法的定义如下：

① 重构 (第 5 章) 是指在不更改代码含义的情况下对其进行改进。

② 契约式设计使用前置条件、后置条件和不变式来回答这样的问题：程序期望什么？保证什么？维持什么？

③ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

在开发计算机系统时使用的形式化方法是描述系统特性的基于数学的技术。这种形式化方法提供了框架，在这些框架内，人们能够以系统的方式而不是随意的方式来说明、开发和验证系统。

形式化规格说明的期望特性是所有规格说明方法的目标。然而，用于形式化方法的基于数学的规格说明语言增加了实现这些理想的可能性。规格说明语言的形式化语法（附录 3）使得我们能够唯一地解释需求或设计，从而排除了解释自然语言（如英语）或图形表示（如 UML）时经常产生的歧义性。集合论和逻辑符号的描述工具使得我们可以清晰地陈述需求。要达到一致，在规格说明中的某个地方陈述的需求就不能与其他地方陈述的需求相矛盾。一致性是通过数学证明将初始事实形式化地映射（使用推理规则）到后面的规格说明中的陈述来保证的^①。

关键点 形式化规格说明应该具有一致性、完整性，且没有歧义。

为了介绍形式化方法的基本概念，我们考虑几个简单的例子来说明数学规格说明的使用，而不是给读者堆砌太多的数学细节。

例 1：符号表。使用一个程序来维护符号表，此符号表在许多不同类型的应用问题中频繁使用。它由一组没有重复的项构成。图 28-7 给出了一个典型的符号表例子，它表示的是操作系统用来保存系统用户名的表。其他表的例子包括工资管理系统中的职员名字表、网络通信系统中的计算机名称表以及产品运输时间表系统中的目的地表。

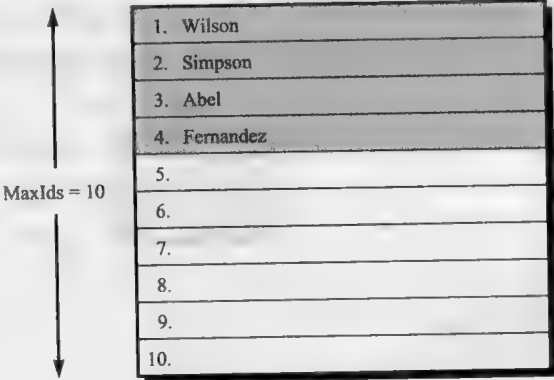


图 28-7 符号表

假设本例中的表包括的职员数量不大于 MaxIds。为表设定限制的这一陈述是条件——被称为数据不变式——的一个构成成分。数据不变式是一个条件，它在包含一组数据的系统的执行过程中总保持为真。上面讨论的符号表的数据不变式有两个构成成分：（1）表中包含的名字数不超过 MaxIds；（2）表中没有重复的名字。在上面描述的符号表程序例子中，这意味着：在系统执行过程中无论什么时候检查符号表，它包含的职员标识符个数总是不超过 MaxIds，并且没有重复。

615

另一个重要的概念是状态。许多形式化语言使用第 11 章所讨论的状态概念；也就是说，系统可能处于多种状态之一，每一种状态都表示外部可观察到的行为模式。然而，在某些规格说明语言中，对术语状态有不同的定义。在这些语言中，系统的状态由系统的存储数据来表示。在符号表程序的例子中使用后面的定义，状态就是符号表。

最后一个概念是操作，这是在系统中发生的读写数据的动作。在符号表程序中如果考虑从符号表加入或去除职员名，则它将关联两个操作：一个操作是将一个指定名增加到符号表，操作 remove() 从符号表中去除一个现有名^②。如果程序提供检查某指定名是否包含在表中的机制，则将有一个返回某种指示值的操作，这个指示值表示名字是否在表中。

① 实际上，即使使用形式化方法，完整性也是难于保证的。在创建规格说明时，系统的某些方面可能还没有定义；某些特性可能被故意遗漏，以允许设计人员具有某些自由度来选择实现方法；最后，在大型、复杂的系统中，不可能考虑每个操作场景。某些事情可能由于疏忽而被遗漏。

② 需要注意的是：在表满状态时不能增加名字，在表空状态时不可能删除名字。

616

有三种类型的条件与操作相关联：不变式、前置条件和后置条件。不变式定义什么保持不变。例如，符号表有一个不变式表示元素的个数总是小于或等于 `MaxIds`。前置条件定义一个特定操作有效的环境。例如，向职员标识符符号表中增加一个名字的前置条件是有效的，仅当表中不含有所要加入的名字，而且在表中只有少于 `MaxIds` 的职员标识符个数时。操作的后置条件定义当操作完成后保证什么为真，这是通过它对数据的影响来定义的。对于 `add()` 操作，后置条件将用数学方法描述表中已经增加了新标识符。

例 2：块处理器。在操作系统中一个更重要的部分是文件子系统，该子系统维护由用户创建的文件。块处理器是文件子系统的一部分。文件存储中的文件由存储设备上的存储块构成，在计算机的操作中，文件的创建和删除需要获取和释放存储块。为了处理这些，文件子系统维持一个未用（空闲）块池，并保持对当前使用块的跟踪。当块从被删除文件释放时，它们通常被加入块队列中以等待进入未用块池。如图 28-8 所示，图中显示了一些部件：未用块池、被操作系统管理的文件使用的块，以及那些等待加入到未用块池中的块。等待块被组织为队列，队列中每个元素包含来自于被删除文件的一组块。

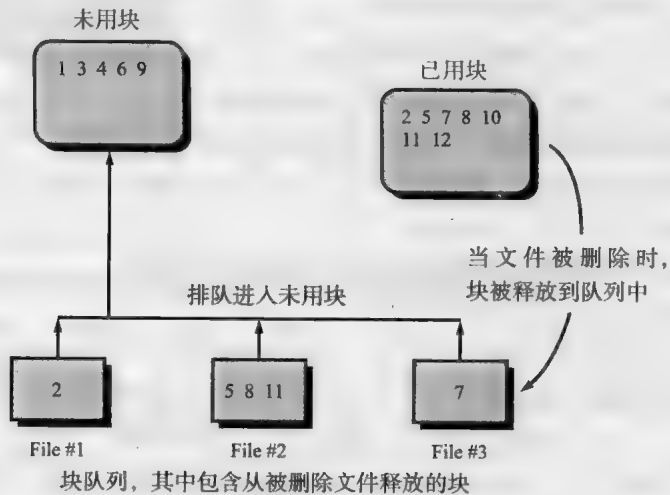


图 28-8 块处理器

对这个子系统来说，状态是空闲块的集合、已用块的集合以及返回块的队列。数据不变式用自然语言表达如下：

617

- 块未同时被标记为未用和已用。
- 所有在队列中的块集合都将是当前已用块集合的子集。
- 队列元素不包含相同的块号。
- 已用块和未用块的集合将是组成文件的块的总和。
- 在未用块集合中没有重复的块号。
- 在已用块集合中没有重复的块号。

与数据相关联的某些操作是：将一个块集合加（`add()`）到队列的末尾，从队列前面去除（`remove()`）一个已用块集合并将其放到未用块集合中，检查（`check()`）块队列是否为空。

`add()` 的前置条件是将被加入的块必须在已用块集合中，后置条件是这个块集合现在处于队列的末尾。`remove()` 的前置条件是队列中必须至少有一项；后置条件是块必须加到未用块集合中。`check()` 操作没有前置条件，这意味着不管状态具有什么值，操作总是有定义的；

后置条件是如果队列为空则返回 true，否则返回 false。

在本节提到的每个例子中，我们已介绍了形式化规格说明的关键概念，但没有重点说明进行形式化规格说明时所需的数学知识。在附录 3 中，我们将考虑如何使用这些数学表示法对系统的某些元素进行形式化说明。

28.7 其他争论

Parnas[Par10] 对典型的程序正确性证明方法提出了一些批评。他认为在实时系统中使用表达式中的变量值来定义计算状态忽略了中断处理的作用。形式化方法通常不鼓励使用表达式的副作用，而这在很多应用领域却是很普遍的。通过使用抽象数据类型便于进行信息隐藏，从而引发状态隐藏，在书写数据不变式时一定会遇到这种情况。这通常使得断言比代码本身更复杂。形式化方法是为确定性程序[⊖]设计的，没有考虑永远运行的程序。

形式化方法依赖于使用试图提供程序简化视图的模型。没有可靠的方法将这些模型转换为程序代码。由于模型或代码有缺陷，因此有可能会证明存在缺陷的程序是正确的。

在形式化方法中，证明软件正确性的过程变成为这样的任务：将程序浓缩成一个合法状态[⊖]的序列或者数据转换的序列。这种做法并不是对于所有程序都是充分的，验证程序运行时行为正确性的断言可能需要用来检查实时算法或并行算法的正确性。

断言可以帮助开发者更好地证明和理解软件，但这仅仅对于小程序是可行的。断言的使用不能排除对于外部文档的需要，由于用于描述系统的自然语言存在的歧义性，外部文档的描述通常是含糊不清的。此外，形式化方法不能帮助软件工程师从多个正确的软件设计方案中进行选择。

28.8 小结

净室软件工程是软件开发的一种形式化方法，它可以生成高质量的软件。它使用盒结构规格说明进行分析和设计建模，并且强调将正确性验证（而不是测试）作为发现和消除错误的主要机制。运用统计使用测试来获取失效率信息，此信息对认证交付软件的可靠性是必需的。

净室方法从使用盒结构表示的分析和设计模型入手，一个“盒”在某特定的抽象层次上封装系统（或系统的某些方面）。黑盒用于表示从外部可以观察到的系统行为。状态盒封装状态数据和操作，清晰盒用于对某状态盒中的数据和操作所蕴含的过程设计建模。

一旦完成了盒结构设计，就可以应用正确性验证。软件构件的过程设计被划分为一系列子函数，为了证明每个子函数的正确性，要为每个子函数定义出口条件，并应用一组子证明。如果每个出口条件均成立，则设计一定是正确的。

一旦完成了正确性验证，便开始统计使用测试。和传统测试不同，净室软件工程并不强调单元或集成测试，而是通过定义一组使用场景并确定对每个场景的使用概率，然后定义符合概率的随机测试来进行软件测试。产生的错误记录和取样、构件和认证模型相结合，使得可以通过数学方法计算软件构件的预计可靠性。

形式化方法使用集合论和逻辑符号描述工具，使得软件工程师能创建事实（需求）的清

⊖ 确定性程序通常开始于定义的开始状态，结束于单一的终结状态。

⊖ 合法状态是指程序变量的集合，可以对程序的输入进行处理，将处理值赋给这些变量。

晰陈述。支配形式化方法的基本概念是：(1) 数据不变式：一个条件表达式，它在包含一组数据的系统的执行过程中总保持为真；(2) 状态：从系统的外部能够观察到的行为模式的一种表示，或者系统访问和修改的存储数据（在 Z 语言或者相关的语言中）；(3) 操作：系统中发生的动作，以及对状态数据的读写。每个操作都与两个条件相关联，即前置条件和后置条件。

净室软件工程或形式化方法是否会广泛使用？答案是“可能不会”。与传统的软件工程方法相比，这两种方法更难于学习，并且对于很多软件工作者是重要的“文化冲击”。但是，下一次你听到某人悲叹“为什么我们不能第一次就将软件做正确”时，你应该知道确实有技术可以做到。

习题与思考题

- 28.1 如果必须选出净室软件工程和传统的或面向对象的软件工程方法根本不同的一个方面，那么这个方面是什么？
- 28.2 增量过程模型和认证如何一同工作以生产出高质量软件？
- 28.3 使用盒结构规格说明，开发 SafeHome 系统的“一遍通过”分析和设计模型。
- 28.4 一个冒泡排序算法定义如下：

```
procedure bubblesort;
var i, t, integer;
begin
  repeat until t=a[1]
    t:=a[1];
    for j:= 2 to n do
      if a[j-1] > a[j] then begin
        t:=a[j-1];
        a[j-1]:=a[j];
        a[j]:=t;
      end
    endrep
  end
```

将该设计划分为子函数，并定义一组条件，从而能够证明该算法是正确的。

- 28.5 为习题 28.4 中讨论的冒泡排序的正确性证明编写文档。
- 28.6 选择你经常使用的一个程序（例如，E-mail 处理器、字处理器、电子表格程序），为此程序创建一组使用场景，定义每个场景的使用概率，然后开发出类似 28.4.1 节中的程序触发和概率分布表。
- 28.7 对习题 28.6 中开发的程序触发和概率分布表，使用一个随机数生成器开发一组用于统计使用测试的测试用例。
- 28.8 用自己的话描述净室软件工程中认证的意图。
- 28.9 你被分配到一个开发传真调制解调器软件的项目组，你的任务是开发应用系统的电话簿部分。电话簿功能可以存储多达 MaxNames 个地址名，相关的公司名、传真号码及其他相关信息。使用自然语言定义：
- 数据不变式。
 - 状态。
 - 可能的操作。
- 28.10 你被分配到一个开发 MemoryDoubler 软件的项目组，此软件为 PC 提供比物理内存看起来更多的内存空间。这项工作通过这样的途径来完成：识别、收集已经分配给已有应用系统但并没有

使用的内存块，并将其进行重新分配。将这些没有使用的内存块重新分配给需要额外内存的应用程序。给出适当的假设条件，并使用自然语言定义：

- a. 数据不变式。
- b. 状态。
- c. 可能的操作。

扩展阅读与信息资源

近年来出版的高级规格说明和验证技术方面的书籍很少。然而，一些新的文献值得考虑。Gabbar (《Modern Formal Methods and Applications》，Springer, 2010) 以及 Boca 和他的同事 (《Formal Methods: State of the Art and New Directions》，Springer, 2010) 编写的书不仅介绍了基础知识和新的发展，还介绍了高级应用。Jackson (《Software Abstractions》，2nd ed., MIT Press, 2012) 介绍了所有基础支持及他所谓的“轻量级形式方法”。Monin 和 Hinchey (《Understanding Formal Methods》，Springer, 2003) 提供了该主题的精彩介绍。Butler 和其他作者 (《Integrated Formal Methods》，Springer, 2002) 介绍了大量形式化方法方面的论文。

除了本章中作为参考文献使用的书目外，Prowell 和他的同事 (《Cleanroom Software Engineering: Technology and Process》，Addison-Wesley, 1999) 提供了对净室方法所有重要方面的深入探讨。Liu (《Formal Engineering for Industrial Software Development: Using the SOFL Method》，Springer, 2010)、Poore 和 Trammell (《Cleanroom Software Engineering: A Reader》，Blackwell Publishing, 1996) 编辑出版的书对净室主题进行了有益的讨论。Becker 和 Whittaker (《Cleanroom Software Engineering Practices》，Idea Group Publishing, 1997) 为那些不熟悉净室方法的人们提供了非常好的概述。

《Cleanroom Pamphlet》(软件技术支持中心, Hill AF Base, April 1995) 包含了许多重要文章的重印版。网络安全与信息系统信息分析中心 (Cyber Security and Information Systems Information Analysis Center, CSIAC, www.thecsiac.com) 提供了净室软件工程方面很多有益的论文、指导书及其他信息源。

通过正确性证明进行的设计验证是净室方法的核心。Cupillari (《The Nuts and Bolts of Proofs》，4th ed., Academic Press, 2012)、Solow (《How to Read and Do Proofs》，5th ed., Wiley, 2009) 以及 Eccles (《An Introduction to Mathematical Reasoning》，Cambridge University Press, 1998) 的书提供了对数学基础的精彩介绍。Stavely (《Toward Zero-Defect Software》，Addison-Wesley, 1998)、Baber (《Error-Free Software》，Wiley, 1991) 以及 Schulmeyer (《Zero Defect Software》，McGraw-Hill, 1990) 的著作详细地讨论了正确性证明。

在形式化方法领域，Hinchey 和 Bowan (《Industrial Strength Formal Methods》，Springer-Verlag, 1999) 以及 Hussmann (《Formal Foundations for Software Engineering Methods》，Springer-Verlag, 1997) 的书提供了有用的指导。本书附录 3 包含了关于形式化方法的数学基础及软件工程中规格说明语言作用的额外讨论。

网上有净室软件工程及形式化方法的大量信息。最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 的“software engineering resources”下找到。

软件配置管理

要点浏览

概念: 开发计算机软件时会发生变更。因为变更的发生,所以需要有效地管理变更。软件配置管理(Software Configuration Management, SCM)也称为变更管理,是一组管理变更的活动。它通过下面的方式来管理变更:识别可能发生变更的工作产品,建立这些工作产品之间的关系,制定管理这些工作产品的不同版本的机制,控制所施加的变更,审核和报告所发生的变更。

人员: 参与软件过程的每个人在某种程度上都参与变更管理,但是有时候也设专人来管理 SCM 过程。

重要性: 如果你不控制变更,那么变更将控制你。这绝不是一件好事。一个未受控制的变更流可以很容易地将一个运行良好的软件项目带入混乱。结果会影响软件质量并且会推迟软件交付。为此,

变更管理是质量管理的重要部分。

步骤: 在构建软件时会创建很多工作产品,因此每个工作产品都需要唯一标识。一旦成功完成标识,就可以建立版本和变更控制机制。为保证在变更发生时维护质量,变更过程需要审核;为了通知那些需要知道变更的人员,需要进行变更报告。

工作产品: 软件配置管理计划(Software Configuration Management Plan)定义变更管理的项目策略。另外,当启动正式的 SCM 时,变更控制过程将产生软件变更请求、报告和工程变更工单。

质量保证措施: 当每个工作产品都可以标识、跟踪和控制时,当每个变更都可以跟踪和分析时,当每个需要知道变更的人员都通知到时,你就做对了。

在开发计算机软件时,变更是不可避免的。而且,对于共同研发项目的软件开发团队来说,变更导致了混乱。如果变更之前没有经过分析,变更实现时没有做相应的记录,没有向需要了解变更的人员报告变更,或没有以一种能够改进质量并减少错误的方式控制变更,都会产生混乱。关于该问题, Babich[Bab86] 是这样说的:

协调软件开发以最大限度地减少混乱的技术称为配置管理。配置管理是对软件开发团队正在构建的软件的修改进行标识、组织和控制的技术,其目标是使错误量减少到最小,并使生产率最高。

软件配置管理(SCM)是在整个软件过程中应用的一种普适性活动。变更可能随时出现,SCM 活动用于:(1)标识变更;(2)控制变更;(3)保证恰当地实施变更;(4)向其他可能的相关人员报告变更。

关键概念

- 基线
- 变更控制
- 配置审核
- 配置管理
- 配置对象元素
- 内容管理
- 标识
- 影响管理
- 中心存储库
- SCM 过程
- 软件配置项
- 状态报告
- 版本控制
- WebApp 和移动 App

明确地区分软件支持和软件配置管理是很重要的。软件支持是一组发生在软件已经交付给客户并投入运行后的软件工程活动。而软件配置管理则是在软件项目开始时就启动,并且只有当软件被淘汰时才终止的一组跟踪和控制活动。

软件工程的主要目标是当发生变更时,使变更更容易地被接受,并减少变更发生时所花费的工作量。本章将探讨使得我们能够管理变更的具体活动。

29.1 软件配置管理概述

软件过程的输出信息可以分成三个主要类别:(1) 计算机程序(源代码和可执行程序);(2) 描述计算机程序的文档(针对不同的软件开发人员和客户);(3) 数据或内容(包含在程序内部和外部)。在软件过程中产生的所有信息项统称为软件配置。

随着软件工程师的工作的进行,软件配置项(Software Configuration Item, SCI)的层次结构就产生了。每一项可以是单个 UML 图一样小或者和完整的设计文档一样大的命名信息元素。如果每个 SCI 只是简单地产生了其他的 SCI,则几乎不会产生混乱。不幸的是,另一个变量进入过程中,就意味着变更。变更可以因为任意理由随时发生。事实上,正如系统工程第一定律(First Law of System Engineering) [Ber80] 所述:不管你处在系统生命周期的什么阶段,系统都可能发生变更,并且在整个生命周期中将会持续不断地提出变更的要求。

这些变更的起因是什么?这个问题的答案就像变更本身一样多变。然而,有 4 种基本的变更源:

- 新的业务或市场条件,引起产品需求或者业务规则的变更。
- 新的客户需要,要求修改信息系统产生的数据、产品提供的功能或基于计算机的系统提供的服务。
- 企业改组或扩大/缩小规模,导致项目优先级或软件工程团队结构的变更。
- 预算或进度的限制,导致系统或产品的重定义。

引述 除变更以外没有什么事情是永恒的。

Heraclitus,
公元前 500 年

提问 请求软件变更的起因是什么?

624

软件配置管理是一组用于在计算机软件的整个生命周期内管理变更的活动。SCM 可被视为应用于整个软件过程的软件质量保证活动。在下面的几节中,我们将描述能够帮助我们管理变更的主要 SCM 任务和重要概念。

29.1.1 SCM 场景[⊖]

典型的配置管理(CM)工作场景包括:负责软件小组的项目经理、负责 CM 规程和方针的配置管理员、负责开发和维护软件产品的软件工程师以及使用软件产品的客户。在本场景中,我们假定由 6 个人组成的团队正在开发一个约 15000 行代码的小型软件。(注意,也可以组建更小或更大团队场景,但是,实际上每个这样的项目都面临着一个问题,就是 CM。)

在操作级别上,SCM 场景包括多种角色和任务。项目经理的职责是保证在确定的时间框架内开发出产品。因此,项目经理必须对软件的开发进展情况进行监控,找出问题,并对问题做出反应。这可通过建立和分析软

提问 每个参与变更管理的人员的职责和应从事的活动是什么?

⊖ 本节摘自 [Dar01]。已经得到卡内基·梅隆大学软件工程研究所的同意,允许翻印由 Susan Dart [Dar01] 编写的“Spectrum of Functionality in CM Systems” (©2001 by Carnegie Mellon University)。

件系统状态报告并执行对系统的评审来完成。

配置管理员的职责不仅是保证代码的创建、变更和测试要遵循相应的规程和方针，还要使项目的相关信息容易得到。为了实现维护代码变更控制的技术，配置管理员可以引入正式的变更请求机制、变更评估机制（通过负责批准软件系统变更的变更控制委员会）和变更批准机制。配置管理员要为工程师们创建和分发任务单，并且还要创建项目的基本环境，而且，还要收集软件系统各个构件的统计信息，比如能够决定系统中哪些构件有问题的信息。

软件工程师的目标是高效地工作。也就是说，软件工程师在代码的创建和测试以及编写支持文档时不做不必要的相互交流；但同时，软件工程师们又要尽可能地进行有效的沟通和协调。特别是，软件工程师可以使用相应的工具来协助开发一致的软件产品；软件工程师之间可以通过相互通报任务要求和任务完成情况进行沟通和协调；通过合并文件，可以使变更在彼此的工作中传播。对于同时有多个变更的构件，要用某种机制来保证具有某种解决冲突和合并变更的方法。依据系统变更原因日志和究竟如何变更的记录，历史资料应该保持对系统中所有构件的演化过程的记录。软件工程师有他们自己创建、变更、测试和集成代码的工作空间。在特定点，可以将代码转变成基线，并从基线做进一步的开发，或生成针对其他目标机的变体。

关键点 一定存在某种机制，能够保证适当地跟踪、管理和执行同一个构件中同时发生的那些变更。

客户只是使用产品。由于产品处于 CM 控制之下，因此，客户要遵守请求变更和指出产品缺陷的正式规程。

理想情况下，在本场景中应用的 CM 系统应该支持所有的角色和任务。也就是说，角色决定了 CM 系统所需的功能。项目经理可以把 CM 看作是一个审核机制；配置管理员可以把 CM 看作控制、跟踪和制定方针的机制；软件工程师可以把 CM 看作变更、构建以及访问控制的机制；而用户可以把 CM 看作质量保证机制。

29.1.2 配置管理系统的元素

在 Susan Dart 关于软件配置管理的内容全面的白皮书 [Dar01] 中，她指明了开发软件配置管理系统时应该具备 4 个重要元素：

- **构件元素**是一组具有文件管理系统（如数据库）功能的工具，使我们能够访问和管理每个软件配置项。
- **过程元素**是一个动作和任务的集合，它为所有参与管理、开发和使用计算机软件的人员定义了变更管理（以及相关活动）的有效方法。
- **构建元素**是一组自动软件构建工具，用以确保装配了正确的有效构件（即正确的版本）集。
- **人员元素**是由实施有效 SCM 的软件团队使用的一组工具和过程特性（包括其他 CM 元素）。

以上这些元素（将在后面几节中详细讨论）并不是相互孤立的。例如，随着软件过程的演化，可能会同时用到构件元素和构建元素；过程元素可以指导多种与 SCM 相关的人员活动，因此也可以将其认为是人员元素。

29.1.3 基线

变更是软件开发中必然会出现的事情。客户希望修改需求，开发者希望修改技术方法，而管理者希望修改项目策略。为什么要修改呢？答案其实十分简单，随着时间的流逝，所有的软件参与者会得到更多的知识（关于他们需要什么、什么方法最好、如何既能完成任务又能赚钱），这些额外的知识是大多数变更发生的推动力，并且造成了很多软件工程实践者难以接受的事实——大多数变更是合理的！

基线是一个软件配置管理概念，它能够帮助我们在不严重阻碍合理变更的条件下控制变更。IEEE（IEEE 标准 610.12-1990）是这样定义基线的：

已经通过正式评审和批准的规格说明或产品，它可以作为进一步开发的基础，并且只有通过正式的变更控制规程才能修改它。

在软件配置项成为基线之前，可以较快地且非正规地进行变更。然而，一旦成为基线，虽然可以进行变更，但是必须应用特定的、正式的规程来评估和验证每次变更。

在软件工程范畴中，基线是软件开发中的里程碑，其标志是在正式技术评审中（第 20 章）已经获得批准的一个或多个软件配置项的交付。例如，某设计模型的元素已经形成文档并通过评审，错误已被发现并得到纠正，一旦该模型的所有部分都经过了评审、纠正和批准，该设计模型就成为基线。任何对程序体系结构（在设计模型中已形成文档）的进一步变更只能在每次变更被评估和批准之后进行。虽然可以在任意细节层次上定义基线，但最常见的软件基线如图 29-1 所示。

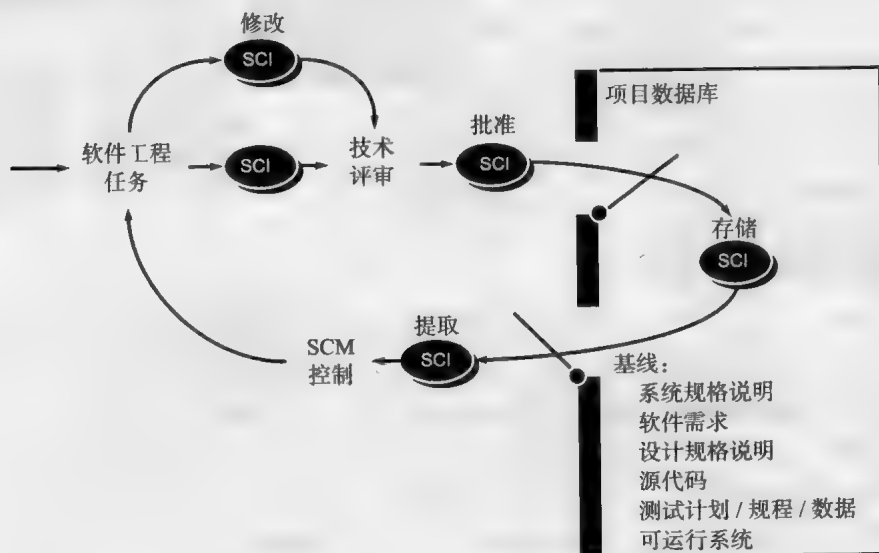


图 29-1 基线化的 SCI 和项目数据库

图 29-1 也说明了基线的形成过程。软件工程任务可能会产生一个或多个 SCI，在这些 SCI 经过评审并被批准之后，就可以将它们放置到项目数据库（也称为项目库或软件中心存储库，见 29.3 节）中。当软件团队中的成员想要修改某个基线 SCI 时，必须将该 SCI 从项目数据库中拷贝到工程师的私有工作区中。但是，这个被提取出的 SCI 只有在遵循 SCM 控制（在本章后面将讨论）的条件下才可以进行修改。图 29-1 中的箭头说明了某个已成为基线的 SCI 的修改路径。

建议 大多数软件变更是合理的，因此没有什么可抱怨的。更确切地说，要确信你已经有恰当控制变更的机制。

626

建议 确保在一个集中的、可控的地点维护项目数据库。

627

29.1.4 软件配置项

软件配置项是在软件工程过程中创建的信息。在极端情况下，大型规格说明中的一节或大型测试用例集中的一个测试用例都可以看作一个 SCI。再实际点，一个 SCI 可以是工作产品的全部或部分（例如，一份文档、一整套测试用例，或者是一个已命名的程序构件）。

除了这些来自软件工作产品的 SCI 之外，很多软件工程组织也将软件工具列入配置管理的范畴，即特定版本的编辑器、编译器、浏览器以及其他自动化工具都被“固化”为软件配置的一部分。因为要使用这些工具来生成文档、源代码和数据，所以当要对软件配置进行变更时，必须得到这些工具。虽然问题并不多见，但一个工具的新版本（如编译器）有可能产生和原版本不同的结果。因此，就像它们协助开发的软件一样，工具也可以基线化为完整配置管理过程的一部分。

在现实中，是将 SCI 组织成配置对象，这些配置对象具有自己的名字，并且按类别存储在项目数据库中。配置对象具有一个名称和多个属性，并通过关系来表示与其他配置对象的“关联”。在图 29-2 中，分别定义了配置对象 DesignSpecification、DataModel、ComponentN、SourceCode 和 TestSpecification。各个对象之间的关系如图中箭头所示，单向箭头表示组合关系，即 DataModel 和 ComponentN 是 DesignSpecification 的组成部分。双向箭头说明对象之间的内在联系，如果 SourceCode 对象发生变更，软件工程师通过查看内在联系能够确定哪些对象（和 SCI）可能受到影响^①。

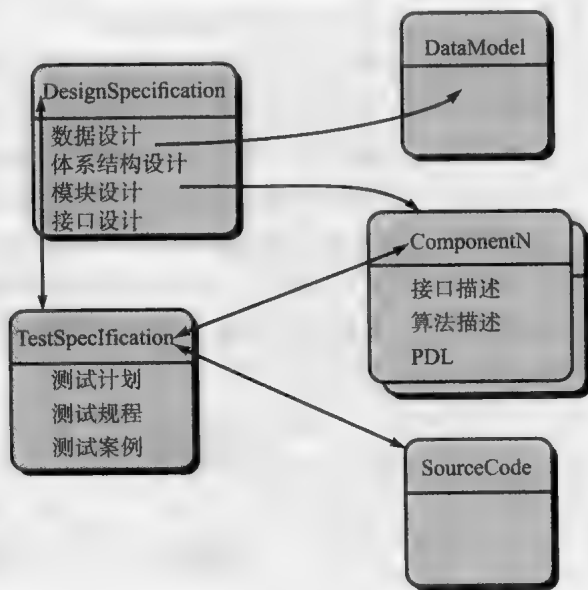


图 29-2 配置对象

29.1.5 依赖性和变更管理

[628]

8.2.6 节介绍了可追溯性内容和可追溯矩阵的应用。可追溯矩阵是记录需求依赖性、架构决策（13.5 节）和缺陷原因（21.6 节）的一种方法。应当这些依赖性需求，当检验到变更时，它将会影响并指导测试用例的选择，还应将其用于回归测试（22.3.2 节）。依赖性管理被看作影响管理^②，这将帮助开发者关注如何改变他们的工作所带来的影响 [Sou08]。

影响分析既关注组织级行为，也关注个人活动，影响管理还包括两项补充内容：（1）确保软件开发者利用策略使他人对自己影响最小化；（2）鼓励软件开发者使用策略使自己对他人影响最小化。注意，最重要的是当一个开发者尝试将他的工作对其他人的影响最小化时，也会降低工作效率，同时其他人必须把对他的工作的影响最小化 [Sou08]。

最重要的是维护软件工作产品，确保开发者在 SCI 中觉察到了依赖性。当对 SCM 存储库进行检入 / 检出时，以及当批准如 29.2 节讨论的变更时，开发者必须建立原则。缺陷跟踪软件也很有用，可以帮助我们找到未发现的 SCI 依赖性。电子化沟通（E-mail、wikis、社交

① 这些关系可以在数据库内定义，数据库（中心存储库）的结构将在 29.3 节讨论。

② 影响管理将在 29.3.4 节讨论。

网络) 为开发者提供便利, 可以分享未记录的依赖性及其引发的问题。

29.2 SCM 中心存储库

SCM 中心存储库是一组机制和数据结构, 它使软件团队可以有效地管理变更。通过保证数据完整性、信息共享和数据集成, 它具有数据库管理系统的一般功能, 此外, SCM 中心存储库还为软件工具的集成提供了中枢, 它是软件过程流的核心。它能够使软件工程项目强制实施统一的结构和格式。

为了实现这些功能, 我们用术语元模型 (meta-model) 来定义中心存储库。元模型决定了在中心存储库中信息如何存储、如何通过工具访问数据、软件工程师如何查看数据、维护数据安全性和完整性的能力如何, 以及将现有模型扩展以适应新需求时的容易程度如何等。

29.2.1 一般特征和内容

中心存储库的特征和内容可以从两个方面来理解: 中心存储库存储什么, 以及中心存储库提供什么特定服务。中心存储库中存储的表示类型、文档和工作产品的详细分类如图 29-3 所示。

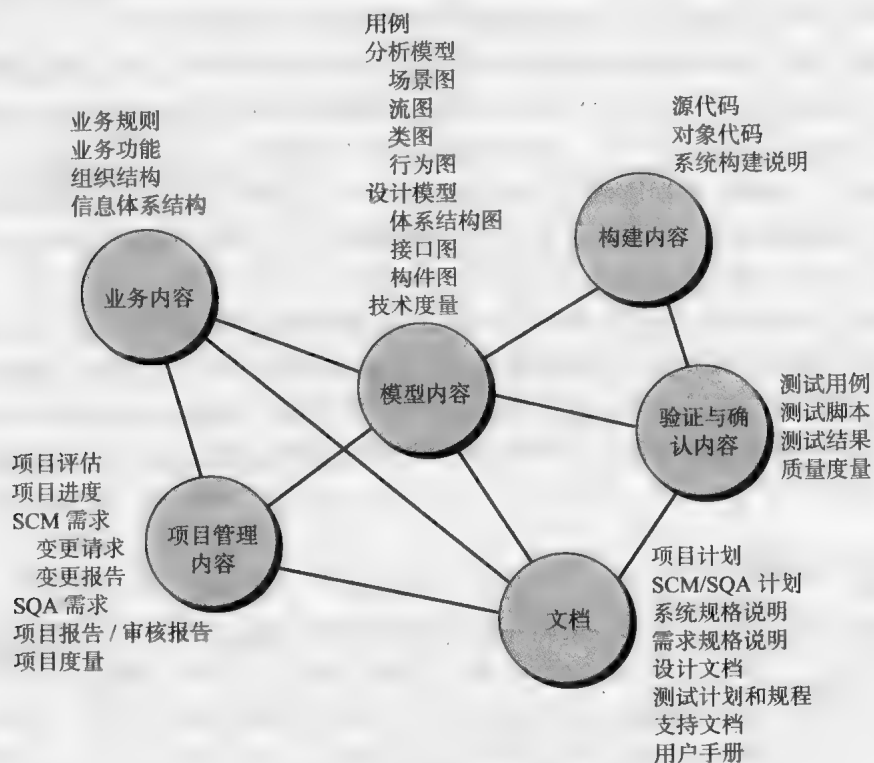


图 29-3 中心存储库的内容

一个健壮的中心存储库能够提供两种不同类型的服务: (1) 期望从任何一个复杂的数据库管理系统得到相同的服务类型; (2) 特定于软件环境的服务类型。

作为软件工程团队的中心存储库, 应该: (1) 集成或直接支持过程管理功能; (2) 支持在中心存储库中管理 SCM 功能的特定规则和维护数据; (3) 提供与其他软件工程工具的接口; (4) 能够存储各种数据对象 (例如,

网络资源 可以通过网址 www.oracle.com/technology/products/repository/in-dex.html 获得商业化的中心存储库。

文本、图形、视频、音频)。

29.2.2 SCM 特征

为了支持 SCM, 中心存储库必须具有支持下列特征的工具集。

版本控制。随着项目的进展, 每个工作产品都可能有很多版本 (29.3.2 节)。中心存储库必须能保存所有版本, 以便有效地管理产品发布, 并允许开发者在测试和调试过程中返回到早先的版本。

中心存储库必须能控制各种类型的对象, 包括文本、图形、位图、复杂文档及一些特殊对象, 如屏幕、报告定义、目标文件、测试数据和结果等。在成熟的中心存储库中可以按任意粒度跟踪对象版本, 例如, 可以跟踪单个数据定义或一组模块。

依赖性跟踪和变更管理。中心存储库要管理所存储的配置对象之间的各种关系。这些关系包括: 企业实体与过程之间的关系、应用系统设计各部分之间的关系、设计构件与企业信息体系结构之间的关系、设计元素与其他可交付工作产品之间的关系等。其中有些仅仅是关联关系, 而有些则是依赖关系或强制关系。

保持追踪这些关系的能力对中心存储库中存储信息的完整性以及基于中心存储库的可交付工作产品的生成是至关重要的, 而且这是中心存储库概念对软件开发过程改进最主要的贡献之一。例如, 如果修改了某 UML 类图, 则中心存储库能够检测出是否有相关联的类、接口描述和代码构件也需要进行修改, 并且能够提醒开发者哪些 SCI 受到了影响。

需求跟踪。这种特殊的功能依赖于相关管理, 并可以跟踪由特定需求规格说明产生的所有设计构件、架构构件以及可交付产品 (正向跟踪)。此外, 还能够用来辨别指定的工作产品是由哪个需求产生的 (反向跟踪)。

配置管理。配置管理设施能够跟踪表示特定项目里程碑或产品发布的一系列配置。

审核跟踪。审核跟踪使我们能够了解变更是在什么时候、由什么原因以及由谁完成等信息。变更的根源信息可以作为中心存储库中特定对象的属性进行存储。每当设计元素进行了修改时, 中心存储库触发机制有助于提示开发人员或创建审核信息条目 (如变更理由) 的工具。

29.3 SCM 过程

软件配置管理过程中定义的一系列任务具有 4 个主要目标: (1) 统一标识软件配置项; (2) 管理一个或多个软件配置项的变更; (3) 便于构建应用系统的不同版本; (4) 在配置随时间演化时, 确保能够保持软件质量。

能够取得上述 4 个目标的过程不应过于原则和抽象, 也不要太繁琐, 这个过程应该具有使软件团队能够解决一系列复杂问题的特色:

- 软件团队应该如何标识软件配置的离散元素?
- 组织应该如何管理程序 (及其文档) 的多个已有版本, 从而使变更能够高效地进行?
- 组织应该如何在软件发布给客户之前和之后控制变更?
- 组织应该如何估算变更影响并有效地管理变更?
- 应该由谁负责批准变更并给变更确定优先级?

关键点 中心存储库必须能维护与软件的许多不同版本相关的 SCI。更重要的是, 它必须提供保证将这些 SCI 组装成一个具体版本配置的机制。

引述 任何变更, 甚至是为了实现更好产品的变更, 也伴随着缺陷和不适。

Arnold Bennett

提问 SCM 过程应该回答什么问题?

- 我们如何保证能够正确地完成变更?
- 应该采用什么机制去评价那些已经发生了的变更?

上述问题引导我们定义了 5 个 SCM 任务: 标识、变更控制、版本控制、配置审核和报告, 如图 29-4 所示。

图中, SCM 任务可以看作一个同心圆的层次结构。软件配置项 (SCI) 在它们的有效生存期内都要从内到外经历各个层次, 最终成为某应用程序或系统的一个或多个版本软件配置的组成部分。当一个 SCI 进入某层时, 该 SCM 过程层次所包含的活动可能适用, 也可能不适用。例如, 在创建一个新的 SCI 时, 必须对其进行标识, 但是, 如果对该 SCI 没有任何变更请求, 那就不必应用

变更控制层的活动, 而直接将该 SCI 指定给软件的特定版本 (版本控制机制开始起作用了)。为了进行配置审核, 要维护 SCI 记录 (SCI 的名称、创建日期、版本标识等), 并将这些记录报告给那些需要知道的人员。下面, 我们将更详细地介绍每个 SCM 过程层次。

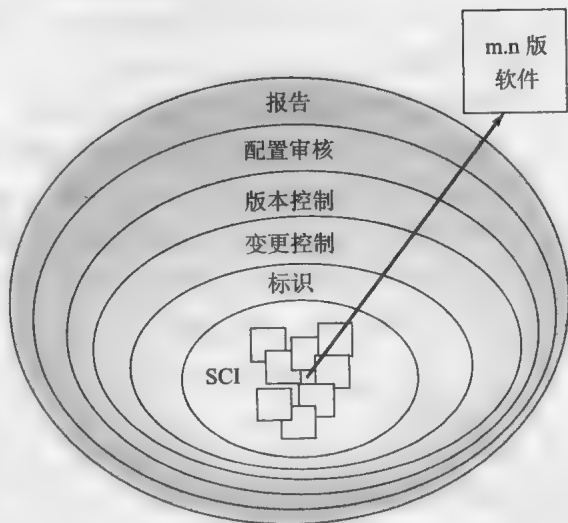


图 29-4 SCM 过程的层次

632

29.3.1 软件配置中的对象标识

为了控制和管理软件配置项, 必须对每个配置项单独命名, 然后用面向对象的方法进行组织。可以进行标识的对象有两种类型 [Cho89]: 基本对象和聚合对象^①。基本对象是软件工程师在分析、设计、编码或测试过程中所创建的“信息单元”。例如, 一个基本对象可以是需求规格说明的一节、设计模型的一个部件、一个构件的源代码或用于测试代码的一组测试用例。聚合对象是基本对象和其他聚合对象的集合。例如, 图 27-2 中的 DesignSpecification 就是一个聚合对象。在概念上, 可将其视为一个已命名的 (已标识的) 指针列表, 指向诸如 ArchitectureModel 和 DataModel 的聚合对象, 以及诸如 ComponentN 和 UMLDiagramN 的基本对象。

每个对象都具有一组能够唯一地标识它的独特特征: 名称、描述、资源表及“实现”。对象名称是能够清楚地标识对象的一个字符串; 对象描述是一个数据项列表, 它能够标识出该对象所表示的 SCI 类型 (例如, 模型元素、程序、数据)、项目标识符以及变更和版本信息; 资源是“对象提供的、处理的、参考的实体, 或者是对象所需要的实体” [Cho89]。例如, 数据类型、特定功能甚至变量名都可以认为是对象资源。“实现”是一个指针, 对于基本对象, 该指针指向其文本单元, 对于聚合对象, 该指针为空。

标识配置对象时也可以考虑各个标识对象之间的关系。例如, 使用下面的简单表示:

类图 <part-of> 需求模型

需求模型 <part-of> 需求规格说明

可以为其创建 SCI 层次结构。

关键点 针对配置对象建立的互联关系可用于评估变更的影响。

633

^① 提出聚合对象的概念是为了将其作为描述软件配置的完整版本而提出的一种机制 [Gus89]。

在很多情况下,在对象层次结构中,对象是跨层次分支而互相关联的。这些交叉的结构关系可以用下面的方式表示:

数据模型 <interrelated> 数据流模型

数据模型 <interrelated> 类 m 的测试用例

第一种相互关系存在于复合对象之间,而第二种相互关系则是存在于聚合对象(DataModel)和基本对象(TestCaseClassM)之间。

在配置对象的标识过程中必须注意到,对象在整个软件过程中是不断演化的。在一个对象被确定为基线之前,它可能会变更很多次,甚至在已经被确定为基线之后,变更也可能会经常发生。

建议 即使项目数据库提供了建立这些关系的功能,但建立起来还是比较耗时,而且很难保持时效性。虽然这些关系对影响分析是很有用的,但是对于整个变更管理不是必需的。

29.3.2 版本控制

版本控制结合了规程和工具,可以用来管理在软件过程中所创建的配置对象的不同版本。版本控制系统实现或者直接集成了4个主要功能:(1)存储所有相关配置对象的项目数据库(中心存储库);(2)存储配置对象所有版本(或能够通过与先前版本间的差异来构建任何一个版本)的版本管理功能;(3)使软件工程师能够收集所有相关配置对象和构造软件特定版本的制作功能;(4)版本控制和变更控制系统通常还有问题跟踪(也叫作错误跟踪)功能,使团队能够记录和跟踪与每个配置对象相关的重要问题的状态。

很多版本控制系统都可以建立变更集——构建软件特定版本所需要的所有变更(针对某些基线配置)的集合。Dart [Dar91]写道:变更集“包含了对全部配置文件的所有变更、变更的理由、由谁完成的变更以及何时进行的变更等详细信息”。

可以为一个应用程序或系统标识很多已命名的变更集,这样就使软件工程师能够通过指定必须应用到基线配置的变更集(按名称)来构建软件的一个版本。为了实现这个功能,就要运用系统建模方法。系统模型包括:(1)一个模板,该模板包含构件的层次结构,以及构建系统时构件的“创建次序”; (2)构建规则; (3)验证规则^①。

在过去几年中,对于版本控制,人们已经提出了很多不同的自动化方法,这些方法的主要区别在于构建系统特定版本和变体属性时的复杂程度以及构建过程的机制。

软件工具 并发版本系统(CVS)

通过工具来实现版本控制对于实现高效变更管理是必要的。并发版本系统(Concurrent Version System, CVS)是在版本控制中普遍使用的工具。它最初是为源代码设计的,但是可运用于任何文本文件。CVS系统的功能有:(1)建立简单的中心存储库;(2)以一个命名文件来维护文件的所有版本,仅仅存储原始文件的各个渐进版本之间的差异;(3)为每位开发

者建立不同的工作目录以实现相互隔离,可避免同时对某个文件进行修改。当开发者完成各自的工作后,由CVS合并处理所有的修改。

要注意的是,CVS不是一个“构建”系统,即它不能构建软件的特定版本。CVS必须集成其他工具(例如Makefile)才能完成这项工作。CVS也不能实现变更控制过程(如变更请求、变更报告和错误

^① 为了评估构件的变更将怎样影响其他构件,可能需要查询该系统模型。

跟踪)。

虽然有上述局限性，但 CVS 仍然“是一个优秀的、开源的、网络透明的版本控制系统，从单个开发者到大型的分散团队都可以使用”[CVS07]。CVS 的客户/服务器体系结构使用户可以从任何有

Internet 连接的地方访问文件，且开源理念使其适用于大部分流行的平台。

可以免费获得 Windows、Mac OS、Linux 和 UNIX 环境下的 CVS，应用的开源版本见 [CVS12]^①。

29.3.3 变更控制

James Bach[Bac98] 很好地总结了现代软件工程范畴内变更控制的真实情况：

变更控制是至关重要的。但是，使变更控制至关重要的驱动力也使它令人厌烦。我们为变更所困扰，因为代码中一个极小的混乱就可能引起产品的失效；但是它也能够修复失效或具有奇妙的新能力。我们为变更所困扰，因为某个喜欢恶作剧的开发者可能破坏整个项目；但是，奇妙的想法也是源自那些喜欢恶作剧的人，而繁重的变更控制过程可能会严重阻碍他们进行创造性的工作。

Bach 认为我们面临的是平衡问题。太多的变更控制会给我们带来问题，太少的变更控制又会给我们带来其他问题。

对于大型的软件工程项目，不受控制的变更会迅速导致混乱。对于这种大型项目，变更控制应该将人为制定的规程与自动工具结合起来。变更控制过程如图 29-5 所示，提交一个变更请求之后，要对其进行多方面的评估：技术指标、潜在副作用、对其他配置对象和系统功能的整体影响，以及变更的预计成本。评估的结果形成变更报告，由变更控制授权人 (Change Control Authority, CCA，对变更的状态及优先级做出最终决策的人或小组) 使用。对每个被批准的变更，需要建立工程变更工单 (Engineering Change Order, ECO)，ECO 描述了将要进行的变更、必须要考虑的约束以及评审和审核的标准。

可以将要进行变更的对象放到一个目录中，该目录只能由实施变更的软件工程师单独控制。完成变更之后，版本控制系统 (如 CVS) 可以更新原始文件。或者，可以将要进行变更的对象从项目数据库 (中心存储库) 中“检出”，进行变更，并应用适当的 SQA 活动，然后，再将对象“检入”到数据库，并应用适当的版本控制机制 (29.3.2 节) 构建该软件的下一个版本。

以上版本控制机制与变更控制过程集成在一起，实现了变更管理的两个主要元素——访问控制和同步控制。访问控制负责管理哪个软件工程师有权限访问和修改某个特定的配置对象；同步控制协助保证两个不同的人员完成的并行变更不会被相互覆盖。

你可能开始对图 29-5 描述的变更控制过程所蕴含的繁文缛节感到不适，这种感觉是很正常的。没有适当的防护措施，变更控制可能会阻碍进展，也可能产生不必要的繁琐手续。大多数拥有变更控制机制的软件开发者 (不幸的是，很多人没有) 通过许多控制环节来避免上面提到的这些问题。

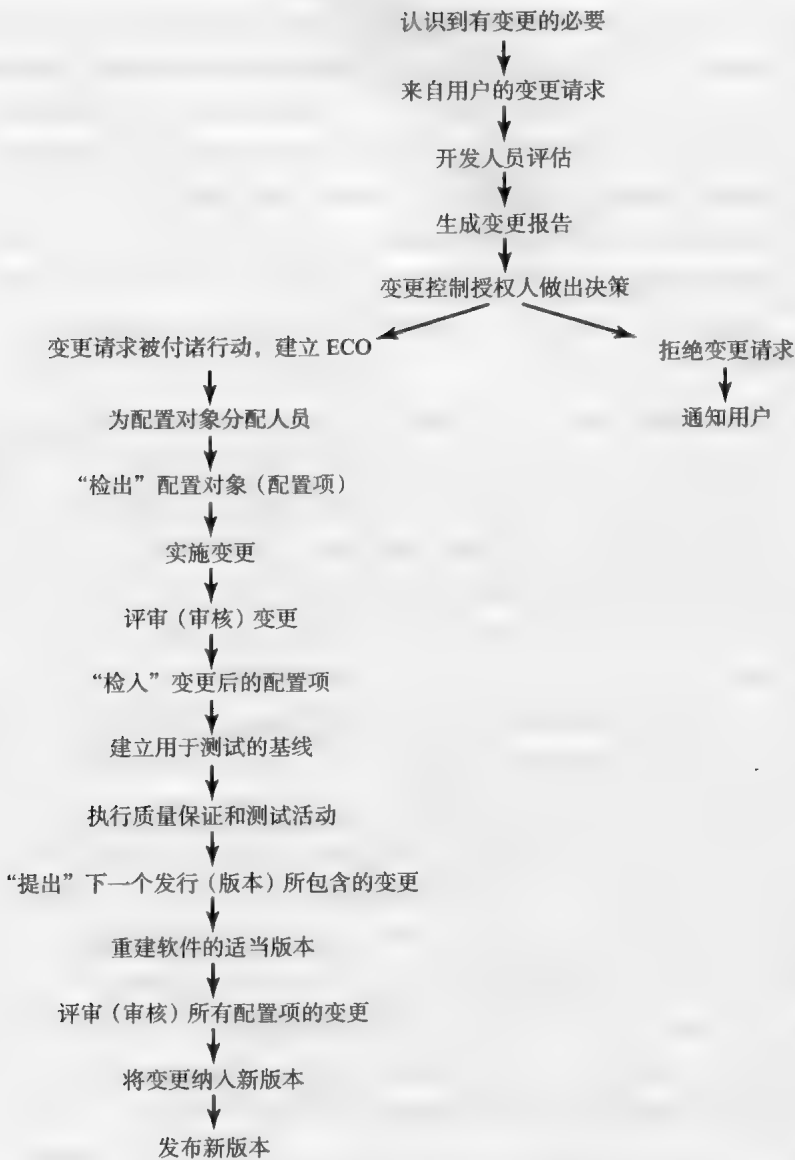
引述 改进的艺术是在变更中保持有序，并且在有序中保持变更。

Alfred North
Whitehead

关键点 应该注意到，许多变更请求可能联合起来形成一个 ECO，而 ECO 通常引起多个配置对象的变更。

635
636

^① 从 <http://olex.openlogic.com/packages/cvs> 下载 [CVS12]。



在 SCI 成为基线之前, 只需要进行非正式的变更控制。还在讨论之中的配置对象 (SCI) 的开发者可以进行任何变更, 只要项目和技术需求证明这些变更是适当的 (只要变更不会影响到开发者工作范围之外的系统需求)。一旦配置对象经过正式技术评审并被批准, 它就会成为基线^①。一旦 SCI 成为基线, 就可以实现项目级变更控制了。这时, 若要进行变更, 开发者必须得到项目管理者的批准 (如果变更是“局部的”), 如果该变更影响到其他 SCI, 则必须得到 CCA 的批准。在某些情况下, 开发者无需生成正式的变更请求、变更报告和 ECO, 但是, 必须对每个变更进行评估, 并对所有的变更进行跟踪和评审。

当交付软件产品给客户时, 正式的变更控制就开始实施了, 正式的变更控制规程如

建议 多选择一些变更控制, 即使你认为不需要这么多。很可能“较多”就是正好的。

^① 也可能因为其他理由创建基线。例如, 当“每日构建”基线创建后, 在给定时间提交的所有构件都变成下一日工作的基线。

图 29-5 所示。

CCA 在控制的第二层和第三层中扮演着主动的角色。CCA 可以是一个人(项目经理),也可以是很多人(例如,来自软件、硬件、数据库工程、支持、市场等方面的代表),这取决于软件项目的规模和性质。CCA 的作用是从全局的观点来评估变更对 SCI 之外事物的影响。变更将对硬件产生什么影响?变更将对性能产生什么影响?变更将怎样改变客户对产品的感觉?变更将对产品的质量和可靠性产生什么影响?还有很多其他问题都需要 CCA 来处理。

引述 变更是不可避免的,自动贩卖机除外。

Bumper sticker

637

SafeHome SCM 问题

[场景] Doug Miller 的办公室, SafeHome 软件项目开始之初。

[人物] Doug Miller (SafeHome 软件团队经理)、Vinod Raman、Jamie Lazar 以及产品软件工程团队的其他成员。

[对话]

Doug: 我知道时间还早,但是我们应该开始讨论变更管理了。

Vinod (笑着说): 很难。今天早上市场部打电话来,有几个需要“重新考虑”的地方。都不是主要的,但这只是刚刚开始。

Jamie: 在以前的项目中,我们的变更管理就非常不正式。

Doug: 我知道,但是这个项目规模更大、更显眼,使我想起……

Vinod (不断点头): 我们已经被“家庭照明控制”项目中不受控制的变更折腾得要命……想起延期就……

Doug (皱着眉): 我不愿意再经历这样的恶梦。

Jamie: 那我们该做什么?

Doug: 在我看来,应该做三件事。第一件,我们必须建立(或借用)一个变更控制过程。

Jamie: 你的意思是如何请求变更吗?

Vinod: 是的。但也包括如何评估变更,如何决定何时进行变更(如果由我们决定),以及如何记录变更所产生的影响。

Doug: 第二件,我们必须获得一个适用于变更控制和版本控制的 SCM 工具。

Jamie: 我们可以为所有的工作产品创建一个数据库。

Vinod: 在这里叫 SCI,绝大部分好用的工具都不同程度地支持这个功能。

Doug: 这是一个良好的开端,现在我们必须……

Jamie: 嗯, Doug, 你说过是三件事的……

Doug (笑着说): 第三件,我们必须使用工具,而且无论如何大家都要遵守变更管理过程。行吗?

29.3.4 影响管理

每次变更发生时,我们都必须考虑软件工作产品的依赖性网络,影响管理(impact management)涉及正确理解这些依赖关系,以及控制它们对其他软件配置项(及其负责人)的影响。

影响管理依靠三种动作来实现[Sou08]。首先,影响网络识别出软件团队(及其他利益相关者)中可能引发变更或受到软件变更影响的人员数目。软件体系结构的清晰定义(第 13 章)对于生成影响网络很有帮助。其次,正向影响管理(forward impact management)评估自身变更对影响网络上的成员的影响,然后告知受到变更影响的成员。最后,反向影响管理

638

(back impact management) 检查其他团队成员所做的变更及其对自己工作的影响, 从而体现减轻影响的机制。

29.3.5 配置审核

标识、版本控制和变更控制帮助软件开发者维持秩序, 否则情况可能将是混乱和不断变化的。然而, 即使最优秀的控制机制也只能在 ECO 建立之后才可以跟踪变更。我们如何能够保证变更的实现是正确的呢? 答案分两个方面: (1) 技术评审; (2) 软件配置审核。

技术评审 (在第 20 章中已经详细讨论过) 关注的是配置对象在修改后的技术正确性。评审者要评估 SCI, 以确定它与其他 SCI 是否一致、是否有遗漏或是否具有潜在的副作用。除了那些非常微不足道的变更之外, 应该对所有变更进行技术评审。

作为技术评审的补充, 软件配置审核针对在评审期间通常不被考虑的特征对配置对象进行评估。软件配置审核要解决以下问题:

1. 在 ECO 中指定的变更已经完成了吗? 引起任何额外的修改了吗?
2. 是否已经进行了技术评审来评估技术正确性?
3. 是否遵循了软件过程, 是否正确地应用了软件工程标准?
4. 在 SCI 中“显著标明”所做的变更了吗? 是否说明了变更日期和变更者? 配置对象的属性反映出该变更了吗?
5. 是否遵循了 SCM 规程中标注变更、记录变更和报告变更的规程?
6. 是否已经正确地更新了所有相关的 SCI?

提问 在配置审核期间需要关注的主要问题是什
么?

在某些情况下, 这些审核问题将作为技术评审的一部分。但是, 当 SCM 是正式活动时, 配置审核将由质量保证小组单独进行。这种正式的配置审核还能够保证将正确的 SCI (按版本) 集成到特定的版本构建中, 并且能够保证所有文档都是最新的, 且与所构建的版本是一致的。

29.3.6 状态报告

配置状态报告 (Configuration Status Reporting, CSR, 有时称为状态账目) 是一项 SCM 任务, 它解答下列问题: (1) 发生了什么事? (2) 谁做的? (3) 什么时候发生的? (4) 会影响其他哪些事情?

配置状态报告的信息流如图 29-5 所示, 每当赋予 SCI 新的标识或更改其标识时, 就会产生一个 CSR 条目; 每当 CCA 批准一个变更 (即创建一个 ECO) 时, 就会产生一个 CSR 条目; 每当进行配置审核时, 其结果要作为 CSR 任务的一部分提出报告。CSR 的结果可以放置到一个联机数据库中或 Web 站点上, 以便软件开发者或维护人员可以按照关键词分类来访问变更信息。此外, 定期生成的 CSR 报告使管理者和开发人员可以评估重要的变更。

建议 为每个配置对象创建一个“须知”列表, 并实时更新。当变更发生时, 保证列表上的每个人都被通知到。

639

软件工具 SCM 支持

[目标] SCM 工具可以支持 29.3 节所讨论的一个或多个过程活动。

[机制] 大部分最新的 SCM 工具都与一个

中心存储库 (一个数据库系统) 相连, 能够提供标识、版本控制和变更控制、审核及报告功能。

[代表性工具]^①

- Software Change Manager。由 Computer Associates (<http://www.ca.com/us/products/Detail/ca-change-manager-enterprise-workbench.aspx>) 发行, 是一个多平台 SCM 系统。
- ClearCase。由 Rational 开发, 提供 SCM 的一系列功能 (<http://www-03.ibm.com/software/products/us/en/clearcase>)。
- Serena Dimensions CMF。由 Serena (<http://www.serena.com/US/products/zmf/index.aspx>) 发行, 提供了适用于传统软件和 WebApp 的全套 SCM 工具。
- Allura。由 SourceForge (<http://sourceforge.net/p/allura/wiki/Allura%20Wiki>) 发行, 提供了版本控制、构建功能、问题/错误跟踪及许多其他管理功能。
- SurroundSCM。由 Seapine Software (www.seapine.com) 开发, 提供完整的变更管理功能。
- Vesta。由 Compac (www.vestasys.org) 发行, 是一个能够支持小型 (<10 KLOC) 和大型 (10 000 KLOC) 项目的没有版权限制的 SCM 系统。

商用 SCM 工具与环境的综合列表可以在 www.cmtoday.com 找到。

29.4 WebApp 和移动 App 的配置管理

在本书的前面部分, 我们讨论了 WebApp 的特殊属性, 以及用来开发 WebApp 和移动 App 的特殊方法^②。WebApp 与传统软件的主要区别就是无处不在的变更。

WebApp 和移动 App 的开发者通常采用迭代和增量过程模型, 用到了很多敏捷软件开发 (第 5 章) 的原理。采用这种方法, 通过客户驱动, 软件工程团队通常可以在很短的时间内开发出 WebApp 的增量。后续开发的增量只是对内容和功能的扩充, 而每个增量都很可能要变更, 这样可以使内容更丰富、使用更容易、界面更美观、导航栏更好、性能更高、安全性更强。因此, 在 WebApp 和移动 App 的敏捷开发中, 变更是有所不同的。

如果你是 WebApp 和移动 App 软件开发团队中的一员, 那么你一定会计与变更打交道, 但是一般的敏捷团队都回避那些过程复杂、过于原则和抽象以及形式化的东西, 而人们通常认为 (虽然不确切) 软件配置管理就具有这些特征。解决这个矛盾并不是要否定 SCM 原则、方法和工具, 而是要使它们满足 WebApp 项目的特定要求。

29.4.1 主要问题

WebApp 和移动 App 对企业的生存和发展越来越重要, 对配置管理的需求也在增长。原因是什么呢? 因为, 如果没有有效的控制, 对这些 App 实施了不适当的修改 (想想看主要特征就是即时性和持续演化), 那么将导致以下问题: 未经许可就发布新产品信息; 错误的或缺乏测试的功能可能会阻碍用户; 安全漏洞可能会危害企业内部系统; 还可能引起其他经济上的不满或更严重的后果。

本章已介绍的软件配置管理 (SCM) 一般策略是适用的, 但必须对这些策略和工具进行

提问 不受控制的变更对 WebApp 有什么影响?

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

② 文献 [Pre08] 有关于 Web 工程方法的综合介绍。

修改,以适应 WebApp 和移动 App 的独特环境。

在制定 WebApp 配置管理策略时应该考虑四个问题 [Dar99]。

内容。典型的 WebApp 包括很多内容——文本、图形、小型程序 (applet)、脚本、音频/视频文件、表单、动态页面元素、表格、流数据等。难点在于如何将这内容组织为一组合理的配置对象 (29.1.4 节),然后再为这些对象建立合适的配置控制机制。一种方法是使用传统的数据建模技术 [Wiki12] 对 WebApp 内容进行建模,并为每个对象附上特殊的属性集。为了建立有效的 SCM 方法,就需要每个对象的动态/静态特性和预期寿命 (如临时的、特定时长或永久对象) 等属性。例如,如果内容项每小时变更一次,则是临时的。与应用到表单构件这种永久对象的控制机制相比,这种项的控制机制是不同的 (不太正式)。

641

人员。因为绝大部分 WebApp 仍然是以独特的方式开发的,所以任何与 WebApp 相关的人员都可以 (且通常可以) 创建内容。而多数内容创建者并没有软件工程知识,根本就不知道还需要配置管理,最终导致应用的增长和变更处于不受控状态。

可伸缩性。适用于小型 WebApp 的技术和控制并不能随规模按比例向上伸展。当将现有的信息系统、数据库、数据仓库以及门户网站互相联系起来时,显然一个简单的 WebApp 会显著增长。随着规模和复杂度的增长,小的变更可能具有深远和无意识的影响,这种影响可能是有问题的。因此,配置控制机制的严格程度应该与 App 规模成正比。

策略。谁“拥有”WebApp? 不管是大公司还是小公司都在争论这个问题,而这个问题的答案对管理和控制活动具有重大的影响。通过以下这些问题 [Dar99] 帮助软件团队理解与 Web 工程相关的策略:谁负责保证 Web 站点上信息的正确性? 在发布信息之前,谁能够保证遵循了质量控制过程? 由谁负责完成变更? 由谁承担变更的成本? 解答上述问题有助于在组织机构中确定负责 WebApp 配置管理过程的人员。

提问 如何确定负责 WebApp 配置管理的人员?

移动 App 的 SCM 技术采纳了许多与敏捷软件开发相同的原则。除了本章前面讨论的传统 SCM 任务,移动 App 的变更也必须考虑每项变更所牵连的安全性和在各种平台环境中对广大用户的影响。

App 商店的戏剧性成长改变了移动软件的实施方式。某个应用的变化仅在数日之后就能广泛传播。因此,在移动 App 商店中发布新版本之前,需要谨慎分析跨平台的影响。

在很多实例中,对 WebApp 和某些移动 App 而言,传统的 SCM 过程可能过于繁琐,但是,在过去几年中,已经出现了为这些应用领域设计的新一代专业化内容管理工具。这些工具建立了获取现有信息 (内容对象) 和管理对象变更的过程,该过程以一种让最终用户可见的方式构建,然后才在客户端环境中显示。

29.4.2 配置对象

WebApp 和移动 App 包括很多配置对象:内容对象 (例如,文本、图形、图像、视频、音频)、功能构件 (例如,脚本、小型应用程序) 和接口对象 (例如,WebApp 的 COM 或 CORBA)。可以按任何方式来标识对象 (指定文件名),只要适用于组织就可行。

642

所有内容都有形式和结构。内部文件的形式是由存储内容的计算机环境所决定的。而表现形式 (常称为显示形式) 是由美学风格和为 WebApp 和移动 App 所建立的设计规则决定的。内容结构定义内容体系结构,即内容结构确定了装配内容对象的方法,从而能给最终用户呈现出有意义的信息。Boiko [Bio04] 将结构定义为“覆盖一组内容块 (对象) 的图,用来

将这些内容组织起来，使需要的人可以访问到这些内容”。

29.4.3 内容管理

在某种意义上，内容管理和配置管理是相关的，因为内容管理系统（Content Management System, CMS）确定了如何（从大量 WebApp 配置对象中）获取现有内容、如何按照能够提交给最终用户的方式构造现有内容，以及在客户端环境下显示这些内容的过程（有适当的工具支持）。

在创建动态应用时最常用到内容管理系统。动态 WebApp 或移动 App 能够“动态地”创建页面。也就是说，由用户向 App 请求特定信息。App 查询服务器端的数据库并形成响应信息，然后提交给用户。例如，某音乐商店（例如，Apple iTunes）提供了成百上千的音轨用于销售。当用户查询一个音轨时，可以查询数据库并获得有关该艺术家、CD（例如，它的封面图片或图形）、音乐内容及音乐试听片段等信息，并配置到标准的内容模板中。最终页面是在服务器端创建的，然后传送到客户端，由最终用户进行验证。比较有代表性的 WebApp 内容管理系统如图 29-6 所示。

引述 内容管理是解决当今信息风暴的一剂良药。
Bob Boiko

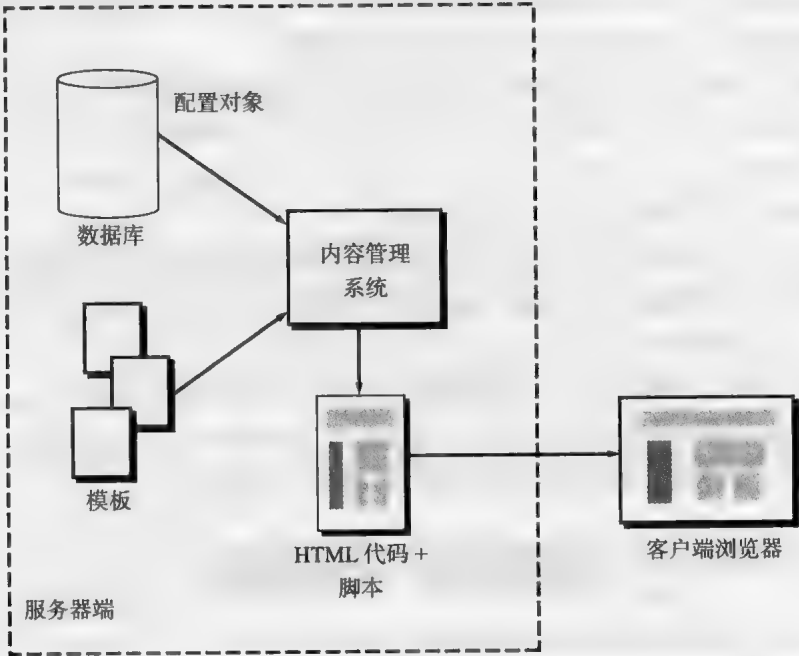


图 29-6 内容管理系统

一般来说，CMS 通过调用三个集成的子系统为最终用户“配置”内容，这三个子系统是收集子系统、管理子系统和发布子系统 [Boi04]。

收集子系统。内容指的是内容开发者必须创建和获取的数据和信息。收集子系统包含用来创建和获取内容的所有活动以及必要的技术，能够实现：（1）将内容转变成标记语言（例如，HTML、XML）可以表示的形式；（2）将内容组织成可以在客户端有效显示的信息包。

内容创建和获取（经常称为创作）通常是和其他开发活动并行出现的，并且经常是由非技术开发人员处理。该活动结合了创新和研究的元素，并

关键点 收集子系统包含以下活动：内容创建、获取，并将其转换为能在客户端表示的形式。

且通过工具支持使得内容作者可以在 WebApp 内部以标准化的方式刻画内容。

一旦有了内容,就要对其进行转换以满足 CMS 的需求。这就意味着从原始内容中剥离不必要的信息(例如,冗余的图形表示)、改变内容格式以满足该 CMS 的需求,并将结果映射到一个能够管理和发布的信息结构中。

管理子系统。收集到内容之后,就必须将其分类保存到中心存储库中,以备随后的获取和使用,而且还要对它进行标注,以确定:(1)当前状态(例如,内容对象是已经实现还是正在开发);(2)内容对象的正确版本;(3)相关的内容对象。因此,管理子系统实现了包含下列元素的中心存储库:

- 内容数据库。为存储所有内容对象所创建的信息结构。
- 数据库功能。使 CMS 能够实现以下功能:查找特定的内容对象(或对象的种类),保存和检索内容对象,以及管理为内容所创建的文件结构。
- 配置管理功能。支持内容对象标识、版本控制、变更管理、变更审核和报告的功能元素和相关工作流。

关键点 管理子系统实现了一个针对所有内容的中心存储库。配置管理在该子系统内执行。

644

除了上述元素之外,管理子系统还实现了管理功能,包括对元数据的管理,以及对控制整个内容结构及支持方式的规则的管理。

发布子系统。必须从中心存储库中提取内容,将内容转换为适于发布的形式,然后进行格式化以便传送到客户端浏览器。发布子系统通过一系列模板来完成这些任务。每个模板对应一个功能,每个功能都可以使用下面三个元素(构件)之一来构造一次发布 [Boi02]:

- 静态元素。不需要进一步处理的文本、图形、媒体和脚本可以直接传送到客户端。
- 发布服务。调用特殊的检索服务和格式化服务功能来定制所需内容(使用预先制定的规则),完成数据转换,以及创建适当的导航链接。
- 外部服务。访问外部的企业信息基础设施,如企业数据或“内部”App。

包含以上三个子系统的内容管理系统可以适用于大多数的 WebApp 项目。

但是,CMS 的基本理论和功能适用于所有的动态 WebApp。

关键点 发布子系统从中心存储库抽取内容,并将其发送到客户端浏览器。

软件工具 内容管理

[目标]辅助软件工程师和内容开发者管理 WebApp 的内容。

[机制]这类工具使 Web 工程师和内容提供者能够以受控的方式更新 WebApp 的内容。大部分工具都建立了简单的文件管理系统,可以按页面进行更新,并对各类 WebApp 内容进行编辑。一些工具还具有版本控制功能,可以获得历史上的早期内

容版本。

[代表性工具]^①

- Open Text Web Experience Management。由 Vignette (<http://www.opentext.com/global/products/web-content-management/web-experience-management/opentext-web-experience-management.htm>) 开发,是一套企业内容管理工具。

① 这里提到的工具只是此类工具的例子,并不代表本书支持采用这些工具。在大多数情况下,工具名称被各自的开发者注册为商标。

- ektron-CMS300。由 ektron (www.ektron.com) 开发, 既提供内容管理功能, 也提供 Web 开发工具。
- OmniUpdate。由 WebsiteASP, Inc.(www.omniupdate.com) 开发, 是一个允许授权的内容提供者在受控的条件下对特定的 WebApp 内容进行更新的工具。
- 有关 SCM 和针对 Web 工程的内容管理工具的其他信息见下列 Web 站点:
 - WebDeveloper (www.webdeveloper.com)。
 - Developer Shed (www.devshed.com)。
 - Webknowhow.net (www.webknowhow.net)。
 - WebReference (www.webreference.com)。

645

29.4.4 变更管理

传统软件变更控制的工作过程 (29.3.3 节) 对于 WebApp 和移动 App 开发来说通常太冗长了。按照大部分 WebApp 和移动 App 开发项目所接受的敏捷方式来完成变更请求、变更报告以及工程变更工单的顺序是不太可能的。那么我们该如何管理对内容和功能方面所提出的连续不断的变更请求呢?

WebApp 和移动 App 开发中一直坚守的信念是“编码并马上运行”, 为了实现此信念下的高效变更管理, 就必须修改常规的变更控制过程。可以将每个变更归为以下 4 种类型中的一种:

- I 类——纠正了一个错误或增加了局部内容 / 功能的内容 / 功能变更。
- II 类——对其他内容对象或功能构件具有影响的内容或功能变更。
- III 类——对整个应用具有重大影响的内容或功能变更 (例如, 主要功能的扩充, 重要内容的增加或减少, 导航中必需的重要变更)。
- IV 类——使一类或多类用户能够立即注意到的重要设计变更 (例如, 界面设计或导航方法的变更)。

对请求的变更进行了分类之后, 就可以按照图 29-7 中所示的算法来处理, 这种方法对于 WebApp 和移动 App 都是适用的。

图中, I 类和 II 类变更可以看作是非正式的, 并且可以按敏捷方式进行处理。对于 I 类变更, 由 Web 工程师评估该变更的影响, 但不需要任何外部的评审或文档。在实施变更时, 配置中心存储库工具只需执行标准的检入和检出过程。对于 II 类变更, 评审该变更对相关对象的影响是 Web 工程师的职责 (也可以要求负责这些对象的开发者进行评审)。如果该变更不会引起对其他对象的大量修改, 则修改时就不需要其他评审和文档。如果需要做大量修改, 就必须做进一步评估和计划。

III 类和 IV 类变更也可以按敏捷方式进行处理, 但是需要一些描述文档和较正式的评审过程。变更描述 (描述变更并对变更所产生的影响进行简要估算) 适用于 III 类变更。将变更描述分发给 Web 工程团队的所有成员, 由这些成员对其进行评审以更好地估算其影响。对于 IV 类变更也要进行变更描述, 但在这种情况下, 是由所有的利益相关者进行评审的。

646

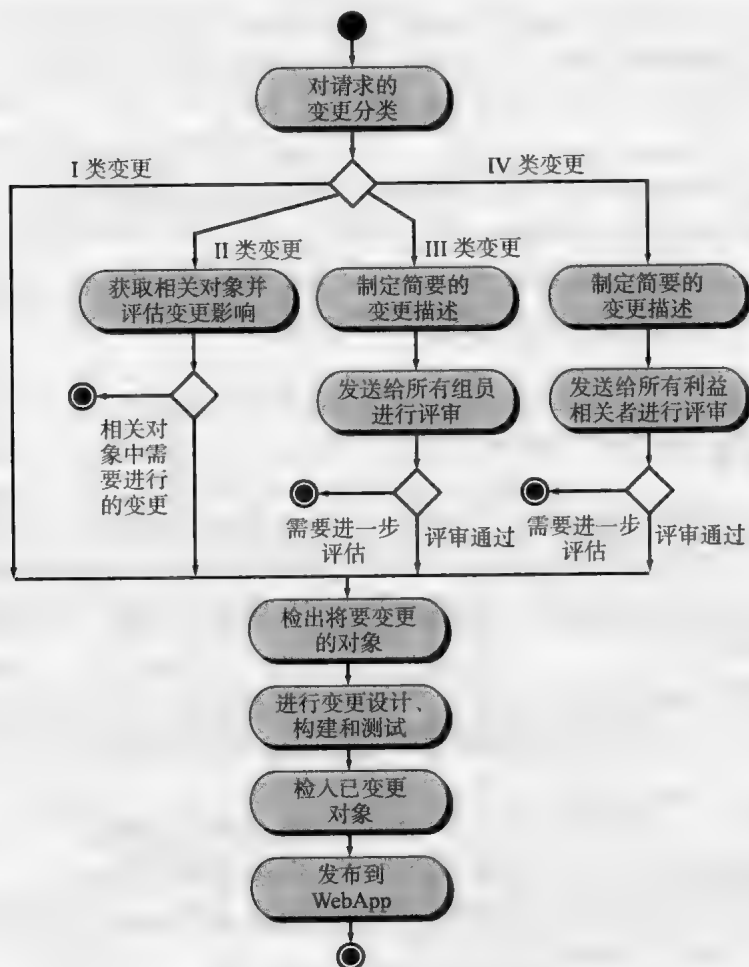


图 29-7 对 WebApp 的变更管理

软件工具

变更管理

[目标] 帮助 Web 工程师和内容开发者对 WebApp 配置对象的变更进行管理。

[机制] 这类工具最初是为传统软件开发的，但是 Web 工程师和内容开发者也可以使用，以控制 WebApp 变更。它们支持自动的检入/检出、版本控制和恢复、变更报告和其他 SCM 功能。

[代表性工具]^①

- Dimension CM。由 Serena (<http://www.serena.com/index.php/en/products/dimensions->

[cm/](http://www.serena.com/index.php/en/products/dimensions-)) 开发，是一套变更管理工具，提供所有 SCM 功能。

- ClearCase。由 Rational (www-03.ibm.com/software/products/us/en/clearcase.html) 开发，是一套提供全部 WebApp 配置管理功能的工具。
- PTC Integrity。由 PTC (<http://www.mks.com/platform/our-product>) 开发，是一种 SCM 工具，能够与所选择的开发环境集成在一起。

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

29.4.5 版本控制

WebApp 和移动 App 的发展过程中要经历很多增量，因此可能同时存在多个不同的版本。最终用户通过 Internet 可以访问某个版本（当前正在运行的应用）；另一个版本（下一个应用增量）可能正处于部署之前测试的最后阶段；而正在开发的第三个版本在内容、界面美观以及功能上都有较大的改变。所以必须清晰地定义配置对象，以便各个配置对象与相应的版本相关联。除此之外，还必须建立控制机制。Dreilinger [Dre99] 认为版本（及变更）控制的重要性如下：

在不受控制的站点中，由于多个创建者具有编辑和上传的权利，因此就会引起潜在的冲突和问题——如果这些创建者在不同时间、不同的办公室工作，就尤为如此。你可能在白天为客户修改了 index.html 文件，但在你完成更改后，下班后在家中工作的另一个开发者，或另一个办公室的开发者，可能在晚上上传了他们自己最新修订的 index.html 文件，这样就无法挽回地完全覆盖了你的工作！

这种情形对于每个软件工程师或者 Web 工程师来说都不陌生。为了避免这种情形，就应该建立版本控制过程。

1. 应该建立 WebApp 和移动 App 项目的中心存储库。中心存储库中将保存所有配置对象（内容、功能构件及其他）的当前版本。
2. 每位 Web 工程师应该创建自己的工作目录。目录中包含了那些在特定时期内正在创建或修改的对象。
3. 所有开发者工作站的时钟应该同步。当两个开发者进行更新的时间非常接近时，这样做可以避免覆盖冲突。
4. 当开发新的配置对象或更改现有的对象时，必须将这些对象存入中心存储库。版本控制工具（见本章前面有关 CVS 的讨论）可以管理来自每位 Web 开发者工作目录的所有检入和检出操作。当中心存储库发生变更时，该工具也可以为所有相关部分提供自动的电子邮件更新。
5. 向中心存储库存入或取出对象时，自动创建带有时间戳的日志信息。这样可以提供有用的审核信息，这些信息还可以作为有效报告的一部分。

版本控制工具能够维护应用的不同版本，如果需要还能够恢复早先的版本。

648

29.4.6 审核和报告

在敏捷开发中，不再强调 WebApp 和移动 App 工作中的审核和报告能^①，但不能两者都被忽视。向中心存储库检入或检出的所有对象都被记录在日志中，任何时刻都可以评审这个日志。还可以创建完整的日志报告，这样 Web 工程团队的所有成员都可以得到指定时间期限内的变更日志。此外，每当向中心存储库检入或检出对象时，还可以自动发送电子邮件进行通知（发给那些感兴趣的开发者和利益相关者）。

① 这是变更的开始。现在人们逐渐强调将 SCM 作为应用安全的一个元素 [Sar06]。通过提供跟踪和报告每个应用对象所做的每个变更的机制，变更管理工具能提供有价值的保护，以防止恶意的变更。

信息栏 SCM 标准

下面的 SCM 标准列表（一部分从 www.12207.com 摘录）是比较全面的：

IEEE 标准	standards.ieee.org/catalog/oils/
IEEE 828	软件配置管理计划
IEEE 1042	软件配置管理
ISO 标准	www.iso.ch/iso/en/ISOOnline.frontpage
ISO 10007-1995	质量管理，CM 指南
ISO/IEC 12207	信息技术—软件生命周期过程
ISO/IEC TR 15271	ISO/IEC 12207 标准实施指南
ISO/IEC TR 15846	软件工程—软件生命周期过程—软件配置管理
EIA 标准	www.eia.org/
EIA 649	国家颁布的配置管理标准
EIA CMB4-1A	数字计算机程序的配置管理定义
EIA CMB4-2	数字计算机程序的配置标识
EIA CMB4-3	计算机软件库
EIA CMB4-4	数字计算机程序的配置变更控制
EIA CMB6-1C	配置和数据管理参考书
EIA CMB6-3	配置标识
EIA CMB6-4	配置控制
EIA CMB6-5	配置状态报告教科书

EIA CMB7-1	配置管理数据的电子交换
美国军用标准	
DoD MIL STD-973	配置管理
MIL-HDBK-61	配置管理指南
其他标准	
DO-178B	航空软件开发准则
ESA PSS-05-09	软件配置管理指南
AECL CE-1001-STD rev.1	安全关键软件的软件工程标准
DOE SCM 检查单	http:// energy.gov/cio/downloads/software-quality-systems-engineering-program-software-configuration-management
BS-6488	英国标准，基于计算机系统的配置管理
最佳实践—UK	政府商业部门： http://www.cabinetoffice.gov.uk/content/office-government-commerce-ogc
CMII	CM 最佳实践学会： www.icmhq.com
配置管理资源指南给那些对 CM 过程和方法感兴趣的人提供了补充信息，见 http://cmpic.com/cmresourceguide.htm 。	

649

29.5 小结

软件配置管理是应用于整个软件过程的普适性活动。SCM 标识、控制、审核和报告修改总是发生在软件开发过程中及交付给客户之后。软件过程中产生的所有信息都应该作为软件配置的一部分，要适当地对配置进行组织，这样才能进行有序的变更控制。

软件配置由一组相关联的对象（也称为软件配置项）构成，这些对象是某些软件工程技术所产生的结果。除了文档、程序和数据外，用于创建软件的开发环境也应该属于配置管

理。应该将所有的 SCI 存放在中心存储库中，中心存储库具有保证数据完整性的机制和数据结构，可以支持其他软件工具，支持软件团队所有成员之间的信息共享，还具有版本控制和变更控制功能。

一旦开发的配置对象通过了评审，它就会成为基线。对基线对象的变更将导致该对象新版本的创建。可以通过分析所有配置对象修订的历史记录来跟踪程序的演化过程。基本对象和复合对象可以形成对象池，通过对象池可以构建不同的版本。版本控制就是管理这些对象的一组规程和工具。

变更控制是一种过程活动，它能够在对配置对象进行变更时保证质量和一致性。变更控制过程从变更请求开始，然后决定是否拒绝该变更请求，最后，对将要变更的 SCI 进行可控制的更新。

配置审核是一种 SQA 活动，它有助于确保进行变更时仍能维护质量。状态报告为那些需要知道变更的人提供了每次变更的信息。

Web 工程的配置管理和传统软件的 SCM 在很多方面是相似的。但是，每个核心的 SCM 任务都应该尽可能简化，而且必须能够达到内容管理的特殊规定。 [650]

习题与思考题

- 29.1 为什么“系统工程第一定律”会成立？变更的主要理由有 4 个，对每个理由都举出几个特例。
- 29.2 实现高效 SCM 系统必需的 4 个元素是什么？简要介绍每个元素。
- 29.3 用自己的话谈谈定义基线的理由。
- 29.4 假定你是某个小项目的负责人，你会为项目定义什么基线？如何控制它们？
- 29.5 设计一个项目数据库（中心存储库）系统，使软件工程师能够存储、交叉引用、跟踪、更新和变更所有重要的软件配置项。数据库应该如何处理同一程序的不同版本？源代码的处理会与文档的处理有所不同吗？两个开发者应该如何避免同时对同一个 SCI 执行不同的修改？
- 29.6 研究某现有的 SCM 工具，然后大概描述它是如何实现版本控制和配置对象控制的。
- 29.7 关系〈 part-of 〉和〈 interrelated 〉表示配置对象之间的简单关系，描述 5 种可能在 SCM 中心存储库中用到的其他关系。
- 29.8 研究某现有的 SCM 工具，并描述它实现版本控制的方法。此外，阅读 2～3 篇有关 SCM 的文章，并描述用于版本控制的不同数据结构和引用机制。
- 29.9 设计一个用在配置审核中的检查表。
- 29.10 SCM 审核和技术评审有什么区别？它们的作用可以归纳为一种评审吗？请说明正反两方面的观点。
- 29.11 简要描述传统软件的 SCM 与 WebApp 的 SCM 之间有何不同。
- 29.12 什么是内容管理？利用网络资源研究内容管理工具的特性，并给出简要的总结。

扩展阅读与信息资源

关于 SCM 的最新资料包括 Aiello 和他的同事 (《 Configuration Management Best Practices: Practical Methods That Work in the Real World 》, Addison-Wesley, 2010)、Moreira (《 Adapting configuration Management for Agile Teams: Balancing Sustainability and Speed 》, Wiley, 2009)、Duvall 和他的同事 (《 Continuous Integration: Improving Software Quality and Reducing Risk 》, Addison-Wesley, 2007)、Leon (《 Software Configuration Management Handbook 》, second edition, Artech House Publishers, 2005)、Moreira (《 The Build Master: Microsoft Software Configuration Management Best

651

Practice》, Addison Wiley, 2005)、Keyes (《Software Configuration Management》, Auerbach, 2004) 以及 Hass (《Configuration Management Principles and Practice》, Addison-Wesley, 2002), 每本书都非常详细地描述了整个 SCM 过程。Moraia (《Software Configuration Management Implementation Roadmap》, Wiley, 2004) 提出了独特的指南, 用以帮助那些在组织内必须实施 SCM 的工程人员。Lyon (《Practical CM III: Best Practices for the 21st Century》, Raven Publishing, 2013, www.configuration.org) 为 CM 专业人员写了一本内容全面的指导书, 书中包含了实现配置管理系统的全方位的实用指导原则 (每年更新)。Girod 和 Shpichko (《IBM Rational ClearCase 7.0: Master the Tools That Monitor, Analyze, and Manage Software Configurations》, Packt, 2011)、Bellagio 和 Mulligan 以一个或多个当今流行的 SCM 工具为例介绍了 SCM。

Berczuk 和 Appleton (《Software Configuration Management Patterns》, Addison-Wesley, 2003) 介绍了有助于理解 SCM 和实现高效 SCM 系统的很多有用模式。Brown 等人 (《Anti-Patterns and Patterns in Software Configuration Management》, Wiley, 1999) 叙述了在 SCM 过程中不能做的那些事情 (反模式), 然后找出它们的解决办法。Humble 和 Fowler (《Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation》, Addison-Wesley, 2010) 以及 Bays (《Software Release Methodology》, Prentice-Hall, 1999) 重点讲述成功产品的发布机制, 这也是对有效 SCM 的重要补充。

由于 WebApp 的动态性, 使内容管理成为 Web 工程师关注的主题。Rockley 和 Cooper (《Managing Enterprise Content: A Unified Content Strategy》, 2nd ed., New Riders, 2012)、Jenkins 和他的同事 (《Managing Content in the Cloud-Enterprise Content Management 2.0》, Open Text Corporation, 2010 和《Managing Content in the Cloud-Enterprise Content Management》, Open Text Corporation, 2005)、White (《The Content Management Handbook》, Curtin University Books, 2005)、Jenkins 和他的同事 (《Enterprise Content Management Methods》, Open Text Cooperation, 2005)、Bioko[Bio04]、Mauthe 和 Thomas (《Professional Content Management Systems》, Wiley, 2004)、Addey 和他的同事 (《Content Management Systems》, Glasshaus, 2003)、Rockly (《Managing Enterprise Content》, New Riders, 2002)、Hackos (《Content Management for Dynamic Web Delivery》, Wiley, 2002) 以及 Nakano (《Web Content Management》, Addison-Wesley, 2001) 介绍了这方面的有价值的内容管理方法。

除了软件配置管理的一般主题外, Halvorson 和 Bach (《Content Strategy for the Web》, 2nd ed., New Riders, 2012)、Hauschildt (《CMS Made Simple 1.6: Beginners' Guide》, Packt, 2010)、Lim 和他的同事 (《Enhance Microsoft Content Management Server with ASP.Net 2.0》, Packt Publisher, 2006)、Ferguson (《Creating Content Management Systems in Java》, Charles River Media, 2006)、IBM Redbook (《IBM Workplace Web Content Management for Portal 5.1》和《IBM Workplace Web Content Management 2.5》, Vivante, 2006)、Fritz 和他的同事 (《Typo3: Enterprise Content Management》, Packt Publishing, 2005) 以及 Forta (《Reality ColdFusion: Intranets and Content Management》, Pearson Education, 2002) 也描述了在使用具体工具和语言环境下的内容管理。

网上有大量关于软件配置管理和内容管理的信息资源。有关软件配置管理的最新参考文献可以在

652

SEPA 网站 www.mhhe.com/pressman 的 “software engineering resources” 下找到。

产品度量

要点浏览

概念: 从本质上讲, 工程是一个量化的学科。产品度量关注软件工作产品具体的、可测量的属性, 帮助软件工程师认识他们所开发的软件的设计和构建。

人员: 软件工程师利用产品度量创建高质量的软件。

重要性: 对计算机软件开发而言, 定性要素总是存在的。但问题是定性评估可能是不够的。软件工程师需要客观标准以帮助指导数据、体系结构、界面和构件的设计。测试人员需要定量指标以帮助选择测试用例及其目标。产品度量为分析、设计、编码和测试能更客观地执行和更定量地评估提供基础。

步骤: 测量过程的第一步是设计适合于所考虑软件的测度和度量。其次, 收集

导出公式化度量所需要的数据。一旦完成计算, 便可基于预先建立的指导原则和过去的数据来分析适当的度量。解释分析结果以获得对软件质量的理解, 且解释的结果将导致需求模型、设计模型、源代码或测试用例的修改。有些情况下, 它也有可能导致软件过程本身的修改。

工作产品: 使用从需求模型和设计模型、源代码及测试用例中收集的数据进行计算而得到产品的度量。

质量保证措施: 应该在开始收集数据之前建立测量的目标, 并以无歧义的方式定义每个产品的度量。只定义几个度量, 然后使用它们去获得对软件工作产品质量的理解。

测量是任何工程过程的一个关键环节。我们使用测量以较好地理解所创建模型的属性, 评估所制造工程产品或系统的质量。但是, 与其他工程学科不同, 软件工程并不是建立在基本的物理定量定律上。直接测量 (例如, 电压、质量、速度或温度) 在软件世界是不常见的。由于软件测量与度量经常是间接得到的, 因此存在公开争论。Fenton[Fen91] 在阐述这个问题时说道:

测量是根据明确的规则来定义, 将数字或符号赋给现实世界实体属性的过程……在物理学、医学、经济学以及近代社会科学中, 我们能够测量原来认为不能测量的属性……当然, 这些测量并不像物理中的一些测量那样精密……但是, 这些测量确实存在 (并且常常根据这些测量做出重要的决策)。我们有责任尝试测量不可测的东西, 以便提高对特殊实体的理解。这在软件工程中与在其他学科中显得同样重要。

但是, 软件业界的一些人始终在争论: 软件是不可测量的, 或者说, 测量的尝试应该推

关键概念

体系结构设计
度量
面向类的度量
功能点 (FP)
目标 / 问题 /
度量 (GQM)
指标
测度
测量
测量原则
面向对象设计的
度量

迟,直到我们对软件以及用于描述软件的属性有较好的理解。这种说法是错误的。

虽然计算机软件产品度量是不完善的,但产品度量为我们提供了一种基于一组明确规定的规则来评价质量的系统方法。同时,产品度量为软件工程师为软件产品提供了现场而不是事后的理解。这使软件工程师能够在潜在问题变成灾难性的错误之前发现和纠正它们。

在本章,我们提出了产品开发时能用于评估产品质量的测度(measure)。这些内部产品属性的测度为软件工程师提供了以下方面的实时指示:需求模型、设计模型及代码模型的功效,测试用例的有效性,以及待开发软件的总体质量。

关键概念

度量的属性

构件级度量

设计度量

移动 App 度量

需求模型度量

源代码度量

测试度量

用户接口设计

度量

WebApp 度量

30.1 产品度量框架

测量将数字或符号赋给现实世界中的实体属性。为达到这个目标,需要一个包含统一规则的测量模型。尽管测量理论(例如,[Kyb84])及其在计算机软件中的应用(例如,[Zus97])等主题超出了本书的讨论范围,但是,为测量软件的产品度量建立一个基本框架和一组基本原则是值得的。

引述 一门科学

与其测量工具一样成熟。

Louis Pasteur

30.1.1 测度、度量和指标

尽管测量(measurement)、测度(measure)和度量(metric)这三个术语常常可以互换使用,但注意到它们之间的细微差别是很重要的。由于“测度”一词可用作名词也可用作动词,因此,这个术语的定义是令人费解的。在软件工程中,“测度”为产品或过程的某些属性的范围、数量、维度、容量或大小提供量化的指标。而“测量”是确定测度的动作。度量在《软件工程术语的 IEEE 标准词汇表》[IEE93b]中的定义为:度量是一个系统、构件或过程具有给定属性的量化测量程度。

提问 测度和测量有什么不同?

当收集了一个数据点时(例如,在一个软件构件中发现的错误数),就已建立了一个测度。收集一个或多个数据点(例如,考察一些构件评审和单元测试,以收集每个单元测试错误数的测度)就产生了测量。软件度量以某种方式(例如,每次评审发现错误的平均数,或每个单元测试所发现错误的平均数)与单个测度相关。

654

软件工程师收集测度并开发度量以便获得指标。一个指标是一个度量或多个度量的组合,它提供了对软件过程、软件项目或产品本身的深入理解。指标提供的理解使项目经理或软件工程师能够调整过程、项目或产品以使其变得更好。

关键点 一个指标就是提供了对

过程、产品或项目深入理解的一个或一些度量。

30.1.2 产品度量的挑战

在过去 40 年中,许多研究人员试图开发出单一的度量值,以为软件复杂性提供全面的测量。Fenton[Fen94]将这种研究描绘为“寻找不可能的圣杯”。尽管已提出了许多复杂性测量[Zus90],但每种方法都对什么是复杂性以及哪些系统属性导致复杂性持有不同的看法。作为类比,我们考虑用来评价汽车吸引力的度量,一些观察者可能强调车身设计,另一些会强调机械特性,还有一些鼓吹价格、性能、燃料经济性或汽车报废时的回收能力。由于这些特性中的任意一个都有可能和其他特性不一致,因此,很难为“吸引力”制定一个单一值。

对计算机软件而言,会出现同样的问题。

然而,仍有必要去测量和控制软件的复杂度。如果一个度量的单一值难以获取,那么可能应该开发针对不同内部程序属性(例如,有效模块化、功能独立性和第12章论及其他属性)的测度。这些测量和由此产生的度量可用作需求模型和设计模型的独立指标。但是新问题再次出现,Fenton[Fen94]说道:“尝试去寻找刻画许多不同属性的测度是危险的,其危险性在于,测度不可避免地必须满足有冲突的目标。这与测量的代表性理论是相反的。”尽管Fenton所说的是正确的,但许多人提出,在软件过程的早期阶段执行的产品测量为软件工程师评估质量[⊖]提供了一致和客观的机制。

网络资源 美国国土安全部汇编了关于产品度量的大量信息,网址是 <http://buildsecuri-tyin.us-cert.gov/bsi/articles/best-practices/measurement.html>。

655

30.1.3 测量原则

在介绍一系列的产品度量之前,重要的是理解基本的测量原则。产品度量的作用主要包括:(1)辅助分析模型和设计模型的评估;(2)提供过程设计和源代码复杂性的指示;(3)方便更有效测试的设计。Roche[Roc94]提出了能用以下5个活动描述的测量过程:公式化、收集、分析、解释、反馈。软件度量仅当被有效地特征化并经过确认以证明它们的价值时才是有用的。人们已经提出了许多度量特征化和确认原则,其中一些有代表性的原则如下[Let03b]:

引述 就像温度测量开始于一根食指……后来逐渐增长到复杂的级别、工具和技术。软件测量的成熟也是如此。

Shari Pfleeger

- 度量应该具有符合要求的数学特性。也就是说,度量的值应该在有意义的范围之内(例如0~1,其中0意味着不存在,1表示最大值,0.5表示中间点)。同时,所谓在合理范围内的度量不应该仅由顺序测量的成分组成。
- 如果度量代表一个软件特征,当正向品质出现时特征值提高,当不理想品质出现时特征值下降,那么度量值提高或降低的方式应当是一致的。
- 每种度量在发布或用于做决策之前,应该在广泛的环境中根据经验加以确认。度量应该测量重要的、与其他因素无关的因素。它应该扩展到大系统中,并能在许多程序设计语言和系统领域中行得通。

尽管公式化、特征化和确认很关键,但收集和分析才是驱动测量过程的活动。Roche[Roc94]为这些活动提供了以下指导原则:(1)只要有可能,数据的收集与分析应能自动化地进行;(2)应该使用有效的统计技术以建立内部产品属性与外部质量特性之间的关系(例如,体系结构的复杂程度是否在产品使用报告中的缺陷数相关);(3)应该为每个度量建立解释性指导原则和建议。

30.1.4 面向目标的软件测量

目标/问题/度量(Goal/Question/Metric, GQM)范型是由Basili和Weiss[Bas84]开发的,这种技术用于为软件过程的任何部分识别出有意义的度量。GQM强调了以下要求:(1)确定特定过程活动的明确测量目标或将要评估的产品特性;(2)定义一组必须回答的问题以达到目标;(3)确定一些良好定义的度量以帮助我们回答这些问题。

656

⊖ 虽然文献中对具体度量的评论是常见的,但许多评论关注深奥的问题而忽略了现实世界中度的主要目标:帮助软件工程师建立一种系统、客观的方法,来获得对其工作的深入理解,并最终提高产品的质量。

目标定义模板 [Bas94] 可用于定义每个测量目标。模板采取以下形式：
在...{ 进行测量的环境 }...环境中，从...{ 对测量感兴趣的人 }...的角度，关于...{ 所考虑的活动或属性 }...方面，为...{ 分析的总体目标[⊖] }...目的，分析...{ 将要测量的属性和活动名 }...。

作为一个例子，考虑 SafeHome 的目标定义模板：

在未来三年要对产品进行改进的环境下，从软件工程师完成工作的角度，关于 SafeHome 具有较强可扩展能力方面，为了评估体系结构构件，分析 SafeHome 软件体系结构。

明确定义了测量目标之后，形成一组问题。回答这些问题有助于软件团队（或其他利益相关者）确定是否已达到测量目标。可能会问到的问题如下：

问题 1：体系结构构件是否将以功能与相关数据分开的方式描述？

问题 2：每个构件的复杂性是限定在一定的范围内以便于修改与扩展吗？

每个问题都应该利用一个或多个测度和度量以量化的方式回答。例如，度量若给出了体系结构构件内聚性（第 12 章）指标，则可能对回答问题 1 有用。本章后面讨论的度量有助于理解问题 2。在每种情况下，所选择（或派生）的度量应该与 30.1.3 节所讨论的测量原则和 30.1.5 节所讨论的测量属性相符。

30.1.5 有效软件度量的属性

尽管已提出了数百种计算机软件度量，但不是所有的度量都为软件工程师提供了实用的支持。一些度量所要求的测量太复杂，一些度量太深奥以致现实世界很少有专业人员能够理解，另一些度量则违背了高质量软件的直观概念。

Ejioogu[Eji91] 定义了一组有效软件度量所应具有的属性。导出的度量及导出度量的测度应该是容易学习的，且其计算不应该需要过多的工作量或时间。度量应该满足工程师对所考虑的产品属性的直观看法（例如，测度模型内聚的度量在数值上应随内聚等级的增长而增长）。度量产生的结果应该总是无歧义的。度量的数学计算应该使用不会导致奇异单位组合的测度。例如，项目组成员使用多种编程语言就会导致可疑的单位组合，这样就没有直观的说服力。度量应该基于需求模型、设计模型或程序结构本身，而不应该依赖于变化多样的编程语言的语法或语义。度量应该提供信息以产生高质量的最终产品。

尽管大多数软件度量满足这些属性，但一些常用的度量可能不满足其中某种属性。例如功能点方法（在 30.2.1 节讨论）——一种软件交付功能的测量。有人提出独立的第三方与其同事就算用相同的软件信息也不可能导出同样的功能点值[⊖]。难道我们应该由此拒绝使用功能点度量吗？当然不能！功能点提供了有用的观点且由此提供了不同的数值，尽管它不能很好地满足某个属性。

网络资源 GQM 的有益讨论可在 www.thecsiac.com/resources/ref_documents/software-acquisition-gold-practice-goal-question-metric-gqm-approach 找到。

建议 经验表明，只有产品度量是直观的且易于计算时，才会得到使用。若需要大量的“计数”，且需要复杂的计算，则该度量不可能得到广泛采纳。

⊖ van Solingen 和 Berghout [Sol99] 建议：目标几乎总是“理解、控制或改善”过程活动或产品属性。

⊖ 可以提出类似的有力的辩论。这是软件度量的本质。

SafeHome 关于产品度量的辩论

[场景] Vinod 的工作间。

[人物] Vinod、Jamie、Ed, SafeHome 软件工程团队的成员, 他们正在进行构件级设计和测试用例设计。

[对话]

Vinod: Doug(Doug Miller, 软件工程经理)告诉我, 我们都应该使用产品度量, 但他有点含糊, 又说他不强迫这件事情, 是否采用由我们自己决定。

Jamie: 那好。我们没有时间做测量工作, 我们都在为维持进度而奋战。

Ed: 我同意 Jamie 的观点。我们的确面临这个问题, 我们没时间。

Vinod: 是的, 我知道, 但是使用产品度量可能会有一些好处。

Jamie: 我并不怀疑这个问题, Vinod, 这是时间问题, 我没有任何剩余时间来做产品度量。

Vinod: 但是, 如果测量能节省你的时间, 那又怎么样呢?

Ed: 你错了, 就像 Jamie 所说的那样, 需

要时间……

Vinod: 不, 等等, 如果它能节省我们的时间, 那又怎么样呢?

Jamie: 你说呢?

Vinod: 返工……正是这样。如果我们使用的一个度量有助于我们避免一个主要的或一个中等的问题, 那它就节省了返工一部分系统所需要的时间, 我们节省了时间。不是吗?

Ed: 我想这有可能, 但你能保证某个产品度量能帮助我们找到问题吗?

Vinod: 你能保证它不能帮助我们找到问题吗?

Jamie: 那你计划怎么办?

Vinod: 我认为我们应该选择一些设计度量, 可能是面向类的, 将它们作为我们所开发的每个评审过程的一部分。

Ed: 我确实不太熟悉面向类的度量。

Vinod: 我花一些时间查查, 然后给一些建议。怎么样, 你们这些家伙?

(Ed 和 Jamie 淡漠地点点头)

658

30.2 需求模型的度量

软件工程的技术工作开始于需求模型的创建阶段。在这个阶段将产生需求并建立设计的基础。因此, 我们非常希望产品度量能提供对分析模型质量的深入理解。

尽管文献中很少出现分析和规格说明度量, 但对项目估算度量进行改造并将其应用于这个环境中是有可能的。这些度量以预测结果系统的规模为目的来研究需求模型。规模有时是(但不总是)设计复杂性的指示器, 而且总是编码、集成和测试工作量增加的指示器。

30.2.1 基于功能的度量

功能点(FP)度量可用作测量系统交付功能的有效手段^①。利用历史数据, 功能点度量可用于: (1) 估算设计、编码和测试软件所需开销或工作量; (2) 预告测试期间将遇到的错误数; (3) 预测实现系统中的构件数和预计的源代码行数。可以利用基于软件信息域的可数(直接)测度和软件

网络资源 关于功能点的更多有用信息可在 ww.w.ifpug.org 和 ww.w.functionpoints.com 找到。

① 关于 FP 度量的书、论文和文章数以百计。一些有价值的文献可以在 [IFP05] 找到。

复杂性评估的经验关系来计算功能点。信息域的值用下列方式定义[⊖]：

外部输入数 (EI)。每个外部输入源于一个用户，或从另一个应用系统中传送过来。它提供了面向不同应用系统的数据或控制信息。输入常用于更新内部逻辑文件 (ILF)。输入应该与独立计数的查询区分开来。

外部输出数 (EO)。每个外部输出从应用系统中导出，并为用户提供信息。在这种情况下，外部输出指的是报告、屏幕、错误消息等。不对报告中的单独数据项进行分开计数。

外部查询数 (EQ)。一个外部查询定义为一个在线输入。其结果是以在线输出 (经常从 ILF 中得到) 的方式产生某个即时软件响应。

内部逻辑文件数 (ILF)。每个内部逻辑文件是驻留在应用系统边界之内的数据逻辑分组，它通过外部输入来维护。

外部接口文件数 (EIF)。每个外部接口文件是驻留在应用系统外部的数据逻辑分组，但它为该应用系统提供有用的信息。

一旦收集了这些数据，就完成了图 30-1 所示的表，且复杂度的值与每个计数相关。利用功能点方法的组织将会制定标准以确定表内某个特定的栏目是简单、中等还是复杂。不过，复杂度的确定毕竟有一定的主观性。

信息域值	计数	加权因子			复杂	
		简单	中等			
外部输入 (EI)	<input type="text"/>	3	4	6	=	<input type="text"/>
外部输出 (EO)	<input type="text"/>	4	5	7	=	<input type="text"/>
外部查询 (EQ)	<input type="text"/>	3	4	6	=	<input type="text"/>
内部逻辑文件 (ILF)	<input type="text"/>	7	10	15	=	<input type="text"/>
外部接口文件 (EIF)	<input type="text"/>	5	7	10	=	<input type="text"/>
总计						<input type="text"/>

图 30-1 计算功能点

利用下面的关系式计算功能点 (FP)：

FP = 总计 × (0.65 + 0.01 × Σ(F_i)) (30.1)

其中，“总计”是从图 30-1 中得到的所有 FP 项的总数。

F_i (i = 1 ~ 14) 是值调整因子 (VAF)，它基于对下列问题的回答来确定 [Lon02]：

- 1. 系统需要可靠的备份和恢复吗？
- 2. 需要专门的数据通信以从应用系统中传输信息或将信息传输到应用系统吗？
- 3. 存在分布处理功能吗？
- 4. 性能是关键的吗？
- 5. 系统将运行在一个现有的、紧张使用的操作环境吗？
- 6. 系统需要在线数据项吗？
- 7. 在线数据项需要对多个屏幕或操作建立输入事务吗？
- 8. ILF 在线更新吗？

关键点 值调整因子用于提供问题复杂度的指标。

⊖ 实际上，信息域值的定义及其计算的方式有点复杂。对于有关的详细内容，有兴趣的读者可以参看 [IFP01]。

9. 输入、输出、文件或查询复杂吗?
10. 内部处理复杂吗?
11. 所设计的代码是可复用的吗?
12. 转换与安装包括在设计中吗?
13. 系统是为不同组织中的多个安装而设计的吗?
14. 应用系统是为了便于变更和易于用户使用而设计的吗?

每个问题可用 0 (不重要或不适用) 到 5 (绝对必需) 间的数值来回答。式 (30.1) 中的常量值和应用于信息域计数的加权因子是由经验确定的。

为说明 FP 度量的使用, 考虑 SafeHome 软件内用户交互功能的一个简单数据流图, 如图 30-2 所示。该功能管理用户交互, 接收一个用户密码来启动或关闭系统, 并且允许对安全区状态和不同安全传感器进行查询。该功能显示了一系列提示信息且将合适的控制信号发送到安全系统的不同构件。

网络资源 一个在线的 FP 计算器可在 http://groups.engin.umd.umich.edu/CIS/course.des/cis375/projects/fp_99/main.html 找到。



图 30-2 SafeHome 用户交互功能的数据流模型

为确定用以计算功能点度量所需的一组关键信息域测度, 需要对数据流图加以评估。图中显示了 3 个外部输入——密码、紧急按钮和启动/关闭, 以及两个外部查询——区域查询和传感器查询。同时给出了一个内部逻辑文件——系统配置文件, 两个外部输出——消息和传感器状态, 4 个外部接口文件——测试传感器、区域设置、启动/关闭和报警。这些数据及相应的复杂度在图 30-3 中给出。

661

信息域值	计数		加权因子				
			简单	中等	复杂		
外部输入 (EI)	3	×	3	4	6	=	9
外部输出 (EO)	2	×	4	5	7	=	8
外部查询 (EQ)	2	×	3	4	6	=	6
内部逻辑文件 (ILF)	1	×	7	10	15	=	7
外部接口文件 (EIF)	4	×	5	7	10	=	20
总计							50

图 30-3 计算功能点

图 30-3 中显示的“总计”必须用式 (30.1) 调整。对于这个例子, 假设 $\sum(F_i)$ 为 46 (一

个中等复杂的产品)。因此,

$$FP = 50 \times (0.65 + (0.01 \times 46)) = 56$$

基于由需求模型中导出的 FP 值,项目组可以估算 SafeHome 软件用户交互功能的整体实现规模。假设过去的的数据表明一个 FP 转换为 60 行代码(使用面向对象语言),每人月生产 12 个功能点。这些历史数据为项目经理提供了重要的计划信息,且这些信息是基于需求模型而不是基于初步估算。进一步假设过去的项目中需求和设计评审期间平均每个功能点有 3 个错误,单元测试和集成测试期间平均每个功能点有 4 个错误。这些数据最终将有助于软件工程师评估其评审和测试活动的完整性。Uemura 和他的同事 [Uem99] 提出,功能点也可以从 UML 类图和时序图中计算出来。

引述 与其在沉思什么是可以应用的“新度量”,不如问问自己较基本的问题:我们将如何处理度量?

Michael Mah,
Larry Putnam

30.2.2 规格说明质量的度量

Davis 和其同事 [Dav93] 提出了用于评估需求模型和相应需求规格说明质量的一系列特征:确定性(无歧义性)、完整性、正确性、可理解性、可验证性、内部与外部一致性、可达性、简洁性、可跟踪性、可修改性、精确性和可复用性。另外,他们注意到,高质量的规格说明是电子存储的、可执行的,或至少是可解释的、对比较重要之处加了注释的,另外还应是稳定的、版本化的、有条理的、附有交叉索引的并且是适度说明的。

关键点 通过测量规格说明的特征,就可能获得量化的确定性和完整性。

尽管在本质上上述许多特征似乎是可以定性的,但每个特征实际上都可以用一个或多个度量来表示。例如,假设在一个规格说明中有 n_r 个需求,则有

$$n_r = n_f + n_{nf}$$

其中, n_f 为功能需求的数量, n_{nf} 为非功能(如性能)需求的数量。

为确定需求的确定性(无歧义性),Davis 等人提出了一种度量,这种度量基于评审者对每项需求解释的一致性:

$$Q_1 = \frac{n_{ui}}{n_r}$$

662 其中, n_{ui} 是所有评审者都有相同解释的需求数目。 Q_1 的值越接近 1,则规格说明的歧义性越低。

功能需求的完整性可以通过计算下面的比率来确定:

$$Q_2 = \frac{n_u}{n_i \times n_s}$$

其中, n_u 为独特功能需求的数目, n_i 是由规格说明明确定义或隐含的输入(激励)的个数, n_s 是所指定状态的个数。 Q_2 测量了已为系统指定的必要功能的百分比。然而,它并没有考虑非功能性需求。为了把这些非功能性需求结合到整体度量中以求完整性,我们必须考虑需求已经被确认的程度:

$$Q_3 = \frac{n_c}{n_c + n_{nv}}$$

其中, n_c 是已经确认为正确的需求个数, n_{nv} 是尚未确认的需求个数。

引述 测量可测量的东西,且使不可测量的东西可测量。

Galileo

30.3 设计模型的度量

很难想象一架新飞机、一个新计算机芯片或一座新办公楼可以在没有定义设计测度、没有确定设计质量各方面的度量的指导下开展设计。然而，基于软件的复杂系统设计通常是在没有测量的情况下进行的。更具有讽刺性的事实是，软件的设计度量是可以得到的，但绝大多数软件工程师却一直不知道它们的存在。

关键点 度量能提供对结构数据和与体系结构设计相关的系统复杂度的深入理解。

与所有其他软件度量一样，计算机软件的设计度量并不完美。关于它们的功效及其应用方式一直存在争论。许多专家提出：在设计测度可以使用之前，需要进一步实验。然而，没有测量的设计是难以接受的选择。

30.3.1 体系结构设计的度量

体系结构设计度量侧重于程序体系结构（第13章）的特征，它强调体系结构和体系结构内模块或构件的有效性。这些度量从某种意义上来讲是“黑盒”的，它并不需要特定软件构件的内部运作知识。

663

Card 与 Glass[Car90] 定义了三种软件设计复杂度测度：结构复杂度、数据复杂度和系统复杂度。

对于层次体系结构（例如，调用-返回体系结构），模块 i 的结构复杂度的定义方式如下：

$$S(i) = f_{\text{out}}^2(i) \quad (30.2)$$

其中， $f_{\text{out}}(i)$ 是模块 i 的扇出数^①。

数据复杂度 $D(i)$ 提供了模块 i 的内部接口复杂度的指示，定义如下：

$$D(i) = \frac{v(i)}{(f_{\text{out}}(i) + 1)} \quad (30.3)$$

其中， $v(i)$ 是传入传出模块 i 的输入和输出变量的个数。

最后，系统复杂度 $C(i)$ 定义为结构复杂度和数据复杂度的总和：

$$C(i) = S(i) + D(i) \quad (30.4)$$

系统的总体体系结构复杂度会随着每个复杂度值的增加而增加。这样集成与测试工作量增加的可能性也较大。

Fenton[Fen91] 提出了一些简单的形态（即外形）度量，使不同的程序体系结构能够用一组简单的尺度进行比较。参考图 30-4 中的调用-返回体系结构，可以定义下述度量：

$$\text{规模} = n + a$$

其中， n 为结点数， a 为弧数。对于图 30-4 所示的体系结构：

$$\text{规模} = 17 + 18 = 35$$

深度 = 4，深度是从根结点到叶结点的最长路径

宽度 = 6，宽度是体系结构任一层次的最多结点数

弧与结点的比例定义为 $r = a / n$ ，它给出了体系结构的连接密度，且对体系结构的耦合性提供了一个简单的指示。对于图 30-4 中的体系结构， $r = 18 / 17 = 1.06$ 。

664

① 扇出数定义为直接隶属于模块 i 的模块数量。也就是直接被模块 i 调用的模块数量。

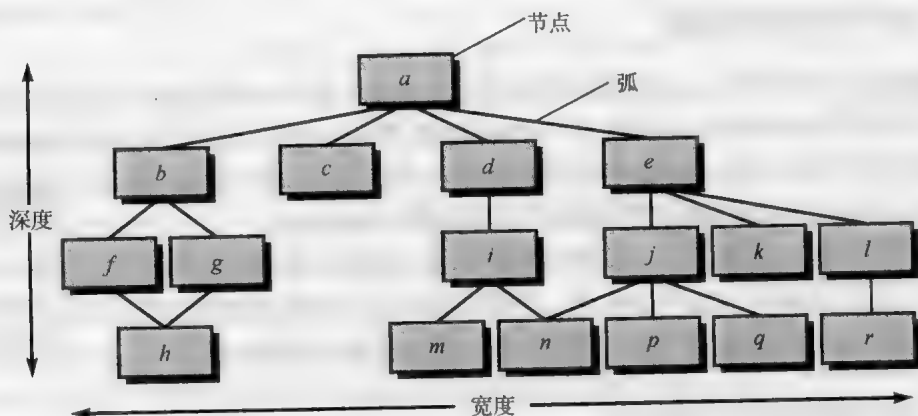


图 30-4 形态度量

美国空军司令部 [USA87] 基于计算机程序可测量的设计特征，开发了一组软件质量指标。利用类似于 IEEE 标准 982.1-2005 [IEE05] 中提出的概念，使用从数据和体系结构设计中取得的信息，导出了一个范围是 0 ~ 1 的设计结构质量指标 (Design Structure Quality Index, DSQI)。为计算 DSQI 的值，必须先清楚下列值 [Cha89]:

S_1 = 程序体系结构中定义的模块总数

S_2 = 模块数，其正确功能依赖于输入数据源或产生用于其他地方的数据 (通常控制模块不计入 S_2 之内)

S_3 = 模块数，其正确功能依赖于前面的处理

S_4 = 数据库中的条目数 (包括所有数据对象及定义对象的所有属性)

S_5 = 数据库不重复的数据项总数

S_6 = 数据库段 (不同记录或个体对象) 的数目

S_7 = 具有单个入口和单出口 (异常处理不看作多重出口) 的模块数

一旦计算机程序的 $S_1 \sim S_7$ 的值确定之后，以下中间值便可以计算出来：

程序结构: D_1 。(若体系结构的设计是用一种独特的方法 (如，面向数据流设计或面向对象设计) 来开发的，则 $D_1=1$ ，否则 $D_1=0$ 。)

$$\text{模块独立性: } D_2 = 1 - \frac{S_2}{S_1}$$

$$\text{模块不依赖于前面处理: } D_3 = 1 - \frac{S_2}{S_1}$$

$$\text{数据库规模: } D_4 = 1 - \frac{S_5}{S_4}$$

$$\text{数据库项的划分: } D_5 = 1 - \frac{S_6}{S_4}$$

$$\text{模块入口/出口特性: } D_6 = 1 - \frac{S_7}{S_1}$$

当这些中间值确定后，DSQI 用下列方式来计算：

$$DSQI = \sum w_i D_i \quad (30.5)$$

其中， i 的值为 1 ~ 6， w_i 是相对权值，考虑了每个中间值的重要性，且 $\sum w_i = 1$ (若所有 D_i 的权值相等，则 $w_i = 0.167$)。

引述 可以将测量看作一条绕行的路，它是必不可少的，因为 (没有量化的支持) 多数人不能做出明确和客观的决策。

Horst Zuse

过去设计的 DSQI 值可以确定下来, 并与目前正在开发的设计相比较。若 DSQI 明显低于平均值, 则意味着需要进一步做设计工作与评审。类似地, 若对现有的设计做重要变更, 则那些变更对 DSQI 的影响可以计算出来。

30.3.2 面向对象设计的度量

关于面向对象设计, 有很多东西是主观的——有经验的设计者“知道”如何去刻画面向对象系统以使之有效地实现客户需求。但是, 当面向对象设计模型的规模和复杂性增加时, 更客观地看待设计特征对有经验的设计者(可获得更为深入的理解)和新手(可获得质量指标, 否则这些指标是得不到的)都是有益的。

在讨论面向对象系统的软件度量时, Whitmire[Whi97]描述了面向对象设计的9个独特的且可测量的特性。规模是通过对面面向对象实体(如类或操作)的静态计数, 结合继承树的深度来测的。复杂性是通过检查面向对象设计的类如何互相关联来看待结构化特征。耦合性是面向对象设计成分间的物理连接(例如, 类间的协作数量或对象间传递的消息数)。充分性是“一个抽象(类)拥有其所需特征的程度及需要具备的特性”[Whi97]。完备性是指类所传递的一系列特性是否能够完全表示问题域。内聚性是检查所有的操作是否能一起工作以达到单一的、明确定义的目标。原始性是指某操作的原子性程度, 即操作不能由包含在类中的其他操作序列构造而成。相似性是两个或多个类在其结构、功能、行为或目的方面的相似程度。易变性测量将发生变更的可能性。

666

实际上, 面向对象系统的产品度量不仅可以应用于设计模型, 也可以应用于分析模型。在下面的几节中, 我们将探讨在类层次和操作层次上提供质量指标的度量。另外, 还要探讨适用于项目管理和测试的度量。

30.3.3 面向类的度量——CK 度量集

类是面向对象系统的基本单位, 因此, 测量和度量个体类、类层次和类协作, 对必须评估设计质量的软件工程师没有多少价值。类封装数据和操作数据的功能。通常的情况是子类(有时称为子女)继承父类的属性和操作, 类常与其他类协作。每种特征都可以用作测量的基础^①。

其中一个被广泛引用的面向对象软件度量集是由 Chidamber 和 Kemerer [Chi94] 提出的。他们提出了6个面向对象系统的基于类的设计度量, 通常称为是 CK 度量套件^②。

每个类的加权方法 (Weighted Methods per Class, WMC)。假定为类 C 的 n 个方法定义的复杂度分别为 c_1, c_2, \dots, c_n , 所选择的特定复杂度度量(例如, 环复杂度)应该规范化, 以便方法的额定复杂度取值为 1.0。

$$WMC = \sum C_i$$

其中, $i=1, \dots, n$ 。方法的数目及其复杂度是实现和测试类所需工作量的合理指标。此外, 方法数目越多, 继承树越复杂(所有的子类继承其父类的方法)。最后, 对于给定类, 随着

① 应该注意到本章讨论的一些度量的有效性问题当前在一些技术文献中还存在争论。那些支持测量理论的人要求某种程度的形式化, 而某些面向对象度量无法提供。然而, 所关注的度量能为软件工程师提供有用的理解也是可接受的。

② Chidamber、Darcy 和 Kemerer 使用术语方法 (method) 而不是操作 (operation)。本节反映了这些术语的使用。

方法数目的增长，它可能越来越特定于应用，由此限制了潜在的复用。因此，WMC 应保持合理的低值。

继承树的深度 (Depth of the Inheritance Tree, DIT)。这个度量为“从结点到树根的最大长度”[Chi94]。参看图 30-5，所显示类层次的 DIT 值为 4。随着 DIT 的增大，低层次的类有可能将继承很多方法。当试图预测类行为时，这将带来潜在困难。较深的类层次 (DIT 值大) 将导致较大的设计复杂性。从正面来讲，较大的 DIT 值意味着许多方法可以复用。

667

子女的数量 (Number Of Children, NOC)。在类层次中，直接从属于某类的子类称为子女。如图 30-5 所示，类 C_2 有 3 个子女——子类 C_{21} 、 C_{22} 和 C_{30} 。随着子女数量的增长，复用也会增加。但是，当 NOC 增大时，如果有些子女不是父类的合适成员，则父类所表示的抽象可能削弱。当 NOC 增大时，测试（需要在其运行环境中检查每个子女）的工作量将也随之增加。

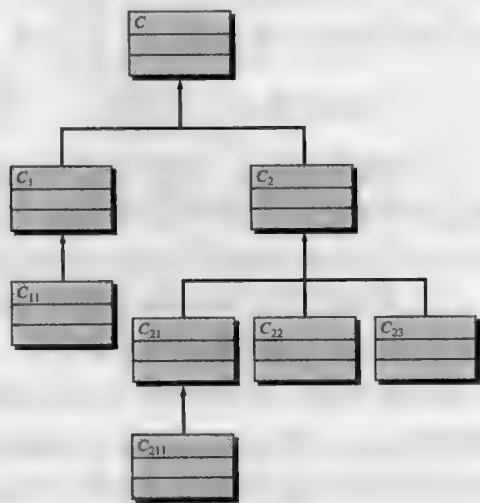


图 30-5 类层次

对象类之间的耦合 (Coupling Between Object classes, CBO)。CRC 模型 (第 10 章) 可用于确定 CBO 的值。本质上，CBO 是在 CRC 索引卡上所列出的类的协作数量^①。当 CBO 增大时，类的可复用性将有可能减小。CBO 的高值也使修改及随之而来的测试变得更为复杂。通常，每个类的 CBO 值应该保持适当的低值。这与传统软件中减少耦合性的一般性指导原则是一致的。

建议 耦合和内聚的概念可以应用到传统软件和面向对象软件。保持类之间的低耦合，以及类和操作的高内聚。

668

对类的响应 (Response For a Class, RFC)。类的响应集是“该类的某对象接收到消息，继而做出响应时可能执行的一组方法”[Chi94]。RFC 是响应集中的方法数。当 RFC 增大时，由于测试序列 (第 24 章) 增加，测试的工作量也随之增加。同样，当 RFC 增大时，类的整体设计复杂性也随之增加。

方法中缺少内聚 (Lack of Cohesion in Method, LCOM)。类 C 中的每个方法访问一个或多个属性 (也称为实例变量)，LCOM 是访问一个或多个相同属性的方法数量^②。若没有方法访问相同的属性，则 $LCOM=0$ 。为说明 $LCOM \neq 0$ 的情形，考虑一个具有 6 个方法的类，其中 4 个方法有一个或多个属性是共同的 (即它们访问共同属性)，因此， $LCOM=4$ 。若 LCOM 的值高，则方法可能通过属性相互耦合，这增加了类设计的复杂性。虽然有些情况下 LCOM 取高值有其理由，但是，我们总是希望保持高内聚，也就是使 LCOM 保持低值^③。

① CRC 索引卡是手工开发的。在可靠地确定 CBO 之前，必须评估其完整性和一致性。

② 该形式化定义有点复杂，详见 [Chi94]。

③ 在一些情况下，LCOM 测度提供了有用的见解，但是在其他一些情形下可能具有误导性。例如，让耦合封装在一个类中在整体上提高了系统的内聚性。因此，至少在这种意义上，较高的 LCOM 确实说明类具有较高的内聚性，而不是较低的内聚性。

SafeHome || CK 度量的应用

[场景] Vinod 的工作间。

[人物] Vinod、Jamie、Shakira 和 Ed，SafeHome 软件工程团队的成员，他们正在进行构件级设计和测试用例设计。

[对话]

Vinod：你们看过我周三发的关于 CK 度量集的描述了吗？做过一些测量吗？

Shakira：不是太复杂。正如你建议的那样，我退回 UML 类和顺序图，并得到 DIT、RFC 和 LCOM 的粗略值。我没找到 CRC 模型，因此不能计算 CBO。

Jamie（笑）：你没找到 CRC 模型是因为它在我这里。

Shakira：这就是我喜欢这个团队的原因——大家能很好地沟通。

Vinod：我做了计算……你们为 CK 度量做过计算吗？

（Jamie 和 Ed 肯定地点头）

Jamie：我有 CRC 卡，所以我看了 CBO。

大部分类看上去相当一致，有一个例外我已经做了标记。

Ed：有些类的 RFC 值相当高，相对于平均值……或许我们应该看看能否对它们进行简化。

Jamie：可能行，也可能不行。我仍然担心时间，我不想修改那些能正常工作的类。

Vinod：我同意这个观点。或许我们应该查找至少有两个或更多的 CK 度量值不太好的类，有两项不利就得改。

Shakira（查看 Ed 的具有较高 RFC 值的类列表）：看这个类，它的 LCOM 和 RFC 的值都高，是两项不利吧？

Vinod：我认为是这样……由于复杂性，实现和测试都是困难的。也许值得设计两个不同的类来实现同样的行为。

Jamie：你认为修改它将节省时间吗？

Vinod：从长远的眼光来看，是这样。

669

30.3.4 面向类的度量——MOOD 度量集

Harrison、Counsell 和 Nithi[Har98b] 提出了一组面向对象设计的度量，这组度量提供了面向对象设计特征的定量指标。以下给出 MOOD 度量的部分样例：

方法继承因子（Method Inheritance Factor, MIF）。一个面向对象系统的类体系结构利用方法（操作）和属性继承的程度可定义为：

$$MIF = \frac{\sum M_i(C_i)}{\sum M_a(C_i)}$$

这里是对 i 从 1 到 TC 求和。TC 定义为体系结构中类的总数， C_i 是体系结构中的一个类，而且，

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

其中：

$M_a(C_i)$ = 可在 C_i 关联中被调用的方法数量

$M_d(C_i)$ = 类 C_i 中声明的方法数量

$M_i(C_i)$ = 类 C_i 中继承的（未被覆写的）方法数量

MIF 的值（属性继承因子（Attribute Inheritance Factor, AIF）以类似的方式定义）指示了继承对面向对象软件的影响。

耦合因子（Coupling Factor, CF）。本章前面提到，耦合是对面向对象设计中元素间连

引述 随着（面向对象）范型的进一步普及，为评估其质量所做的面向对象软件的分析变得越来越重要。

Rachel Harrison
et al.

接的指示。MOOD 度量定义耦合如下：

$$CF = \sum_i \sum_j is_client \frac{(C_i, C_j)}{(TC^2 - TC)}$$

这里，对 i 从 1 到 TC 和 j 从 1 到 TC 求和。函数 $is_client=1$ ，当且仅当用户类 C_i 与服务类 C_j 间存在关系，且 $C_i \neq C_j$ ；否则， $is_client=0$ 。

尽管许多因素都会影响软件复杂性、可理解性和可维护性，但是，可以合理地得出这样的结论：随着 CF 值的增大，面向对象软件的复杂性将随之增加，由此，可理解性和可维护性和潜在的可复用性将受到损害。

Harrison 和他的同事 [Har98b] 对 MIF、CF 及其他度量进行了详细分析，并检查了它们在设计质量评估中的有效性。

30.3.5 Lorenz 和 Kidd 提出的面向对象的度量

在关于面向对象度量的书籍中，Lorenz 和 Kidd[Lor94] 将基于类的度量分为与构件级设计相关的 4 类：规模、继承、内部和外部。对于面向对象设计类，面向规模的度量侧重于对单个类的属性和操作的计数以及面向对象系统的整体平均值。基于继承的度量侧重于整个类层次中操作被复用的方式。类的内部度量考察内聚（30.3.3 节）与面向代码的问题。外部度量检查耦合性与复用。Lorenz 和 Kidd 提出的一个度量的例子如下：

类的规模 (Class Size, CS)。类的整体规模可以用下面的测度来确定：

- 封装在类中的操作总数（包括继承来的和私有的实例操作）。
- 封装在类中的属性总数（包括继承来的和私有的实例属性）。

Chidamber 和 Kemerer 提出的 WMC 度量（30.3.3 节）也是类规模的加权测量。正如前面提到的，大的 CS 值指明类可能有太多的职责。这将减小该类的可复用性且使实现和测试更复杂。一般来讲，在确定类的规模时，继承的或公有的操作与属性应该有较大的加权值 [Lor94]。私有操作和属性允许特化且在设计中更加局部化。也可以计算类属性和操作数量的平均值。规模的平均值越低，类在本系统中能被广泛复用的可能性越高。

建议 在分析模型的评审期间，CRC 索引卡将提供 CS 期望值的合理指标。如果遇到一个类包含大量的职责，则考虑将其分解。

30.3.6 构件级设计的度量

传统软件构件的构件级设计度量侧重于软件构件的内部特性，并包括“3C”的测度——内聚性 (cohesion) [Bie 94]、耦合性 (coupling) [Dha 95] 和复杂度 (complexity) [Zus97]。这些测度有助于软件工程师判断构件级设计的质量。

在需要考虑模块内部运作知识的意义上讲，本节讨论的度量是“玻璃盒”，一旦开发了过程设计 (procedural design)，就可以应用构件级设计度量。另外，它们也可以延迟到有源代码时才用。

关键点 可以计算构件的功能独立性（耦合和内聚）的测度，并用其评估设计的质量。

30.3.7 面向操作的度量

由于类是面向对象系统中最主要的单元，因此，已提出的度量很少是针对类中操作的。Churcher 和 Shepperd[Chu95] 在讨论这个问题时说：“近来的研究表明，在语句数量和逻辑复杂性 [Wil93] 这两个方面，方法都倾向于短小，并提出系统的连接结构可能比单个

模块的内容更重要。”然而，通过检查方法（操作）的平均特征，还是可以获得一些了解。Lorenz 和 Kidd[Lor94] 提出的以下三个简单度量比较适当。

平均操作规模（Average Operation Size, OS_{avg} ）。可以通过对代码行计数或操作发送的消息数来确定规模。当单一操作所发送的消息数增加时，类中的职责有可能未能很好地分配。

操作复杂性（Operation Complexity, OC）。任何一种为传统软件提出的复杂性度量都可以用来计算操作的复杂性 [Zus90]。由于操作应该限于特定的职责，因此设计者应该保持 OC 的值尽可能的低。

每个操作参数的平均数（Average Number of Parameters per Operation, NP_{avg} ）。操作的参数越多，对象间的协作越复杂。一般来讲，应该保持 NP_{avg} 的值尽可能的低。

30.3.8 用户界面设计的度量

尽管在人机界面设计方面有许多重要文献（第15章），但是，有关界面质量和易用性度量的信息却比较少。

Web 页面度量研究 [Ivo01] 显示，布局元素的简单特征也对 GUI 设计的感知质量有重要的影响。在 Web 页面中包含的单词、链接、图形、颜色和字体（连同其他特征）的数量都会影响该页面的感知复杂度和质量。

值得指出的是，尽管 UI 指标可能在某些情况下是有用的，但终裁者应该是基于 GUI 原型的用户输入。Nielsen 和 Levy[Nie94] 谈道：“若一个人仅基于用户观点选择界面（设计），他就有相当大的机会获得成功。用户的平均任务性能和他们对 GUI 的主观满意度是紧密相关的。”

引述 从为洗碗机上添加碗盘这项工作中，你至少能学会一条用户界面设计原则，就是在机器上堆得太多，会使哪个碗盘都洗不干净。

作者不详

30.4 WebApp 和移动 App 的设计度量

WebApp 的有用的测度和度量集为如下问题提供了定量答案：

- 用户接口是否提高了可用性？
- WebApp 的美学设计对于应用领域是否合适？是否能够得到客户的认可？
- 是否能够以最小工作量提供最多信息的方式来设计内容？
- 导航是否有效和直接？
- WebApp 体系结构设计是否适应 WebApp 用户的特定目标和目的、内容结构和功能结构以及有效地使用系统所需的导航流程？
- 构件设计是否减少了流程复杂度且提高了正确性、可靠性和性能？

因为现在还没有能提供量化答案的有效度量集，所有这些问题只能定性地处理。接下来，我们提供在文献中已经提出的关于 WebApp 设计度量和移动 App 系统设计度量的代表性例子。值得注意的是，这些度量还没有被验证，因此应该审慎地使用。

界面度量。对于 WebApp，可以考虑下面的界面度量：

建议 其中许多度量都可应用于所有的用户界面，应该和 30.3.8 节中的度量结合。

建议的度量	描述
布局恰当性	界面上实体的相对位置
布局复杂度	界面定义的不同区域的数量 ^①

① 不同区域指的是一个在布局显示范围内的区域，它能够完成一些具体的相关功能（例如，菜单栏、静态图形显示、内容域、动画显示）。

(续)

建议的度量	描述
布局区域复杂度	每个区域不同链接的平均数
识别复杂度	在进行导航或数据输入之前用户必须查看的不同项的平均数
识别时间	用户为给定任务选择恰当活动的平均时间（单位：秒）
输入工作量	具体功能所需的敲键平均数
鼠标选中工作量	每个功能鼠标选中的平均数
选择复杂度	每个页面能选择链接的平均数
内容获取时间	每个 Web 页面文本单词的平均数
记忆负担	为实现特定目的用户所必须记住的不同数据项的平均数

美学（平面设计）度量。本质上，美学设计依赖于定性的判断，通常不遵守测量和度量的规则。然而，Ivory 及其同事 [Ivo01] 提出的一组测度可能在评估美学设计的影响时有用。该度量集如下：

内容度量。该类度量强调内容复杂度和内容对象的聚集。相关内容可以参考文献 [Men01]。这些度量包括：

建议的度量	描述
单词个数	一个页面中出现的单词总数
主体（Body）文本百分比	主体文本与显示文本（即 Header 部分）的单词百分比
强调主体文本百分比	强调（如粗体、大写）的主体文本的比例
文本定位数量	左对齐文本位置变动次数
文本群数量	用颜色、边框、破折号或列表等强调的文本域
链接数量	页中的总链接数
页面大小	页面总字节数（包括元素、图形、样式表）
图形百分比	页面图形的字节数量的百分比
图形数	页中图形的个数（不包含脚本、applet 和对象中的图形）
颜色数	采用的颜色总数
字体数	采用的字体总数（即字体类型 + 大小 + 粗体 + 斜体）

建议的度量	描述
页面等待	以不同的连接速度下载页面所需的平均时间
页面复杂度	页面使用的不同类型的媒体的平均数量，不包括文本
图形复杂度	每页图形的平均数
音频复杂度	每页音频的平均数
视频复杂度	每页视频的平均数
动画复杂度	每页动画的平均数
扫描图形复杂度	每页扫描图形的平均数

导航度量。该类度量处理导航流程的复杂度 [Men01]。通常来说，它们只可应用于静态的 WebApp，不包含动态产生的链接和页面。

建议的度量	描述
页面链接复杂度	每页链接的数量
连通性	内部链接的总数，不包括动态产生的链接
连通密度	连通性除以页面个数

673

674

利用建议度量的子集可能会得到一些经验关系。根据这些经验关系, WebApp 开发团队可以基于复杂度的预测估计来评估技术质量和预计工作量。该领域的工作还有待将来完成。

软件工具 WebApp 的技术度量

[目标] 帮助 Web 工程人员开发有意义的 WebApp 度量, 这些度量能够衡量应用的总体质量。

[机制] 工具机制多样。

[代表性工具]^①

- Netmechnic Tools。由 Netmechnic(www.netmechnic.com) 开发, 是一组工具的集合。有助于提高网站的性能, 关注具体的实现问题。
- NIST Web Metrics Testbed。由国家标准技术研究所(zing.ncsl.nist.gov/WebTools/) 开发, 包含下面一些可下载的有用工具。
- Web Static Analyzer Tool (WebSAT)——根据典型的可用性原则检查 HTML Web

页面。

- Web Category Analysis Tool (WebCAT)——使工程师构造和管理 Web 分类分析。
- Web Varuable Instrumenter Program (WebVIP)——给网站装备工具, 截获用户交互的日志。
- Framework for Logging Usability Data (FLUD)——实现文件格式化工具和解析器来描述用户交互日志。
- VisVIP Tool——通过网站实现用户导航路径的 3D 可视化。
- TreeDec——将导航辅助加到网站的页面中。

30.5 源代码的度量

Halstead 的“软件科学”理论 [Hal77] 提出了计算机软件的第一个分析“定律”^②。Halstead 利用可以在代码生成后导出的或一旦设计完成之后可以估算得到的一组基本测度, 给出了用于计算机软件开发定量定律。软件科学使用的一组基本测度如下:

n_1 = 在程序中出现的不同操作符的数量

n_2 = 在程序中出现的不同操作数的数量

N_1 = 出现的操作符总数

N_2 = 出现的操作数总数

Halstead 利用这些基本测度开发了一些表达式, 这些表达式可用于度量整个程序的长度、算法的最小潜在信息量、实际信息量(指定一个程序所需的比特数)、程序层次(一种软件复杂性测度)、语言级别(对给定语言为一常量)和其他特征(例如, 开发工作量、开发时间, 甚至软件中的预计缺陷数)。

Halstead 表明, 长度 N 可以估算如下:

引述 人脑遵守一套比较严格的规则(用于开发算法), 这些规则比已知的规则严格得多。

Maurice Halstead

建议 操作符包含所有的控制流结构、条件和数学操作。操作数包含所有的程序变量和常量。

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。

② 应该注意到 Halstead 的“定律”已经引起了很多争议, 许多人认为其基本理论有缺点。然而, 人们已经进行了对已选编程语言的实验验证(例如, [Fel89])。

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

而程序的信息量可以定义为：

$$V = N \log_2 (n_1 + n_2)$$

应该注意到， V 随着编程语言的不同而不同，它表示说明一个程序所需要的信息量（以比特计数）。

理论上，一个特定算法必定存在一个最小信息量。Halstead 将信息量比率 L 定义为程序最简洁形式的信息量与实际程序的信息量之比。实际上， L 一定总是小于 1 的。根据基本测度，信息量比率可以表示为：

$$L = \frac{2}{n_1} \times \frac{n_2}{N_1}$$

Halstead 的工作有必要通过实验验证，而且大量的研究已针对软件科学进行。此项工作的讨论超出了本书的范围，进一步信息参见 [Zus90]、[Fen91] 和 [Zus97]。

30.6 测试的度量

大部分度量都侧重于测试过程而不是测试的技术特征本身。一般来讲，测试者必须依靠分析、设计和代码度量来指导测试用例的设计与执行。

体系结构设计度量提供了与集成测试相关的难易信息（30.3 节）以及对专用测试软件（如桩和驱动）的需求。环复杂度（一种构件级设计度量）是基本路径测试（第 23 章描述的测试用例设计方法）的核心。此外，环复杂度还可用来定位要进行广泛单元测试的候选模块。环复杂度高的模块可能比环复杂度低的模块更易于出错。因此，测试人员应该在将这些模块集成到系统之前花费超过平均值的工作量以发现该模块中的错误。

关键点 测试度量分成两大类：
(1) 预计在不同的测试级别可能需要的测试的数量；
(2) 强调针对给定构件的测试覆盖。

30.6.1 用于测试的 Halstead 度量

从 Halstead 测度（30.5 节）导出的度量也可用来估算测试工作量。利用程序信息量 V 的定义和程序层次 PL，可以计算 Halstead 工作量 e ：

$$PL = \frac{1}{[(n_1/2) \times (N_2/n_2)]} \quad (30.6a)$$

$$e = \frac{V}{PL} \quad (30.6b)$$

676

分配给模块 k 的工作量占整体测试工作量的百分比可以用下式进行估算：

$$\text{测试工作量百分比}(k) = \frac{e(k)}{\sum e(i)} \quad (30.7)$$

其中， $e(k)$ 是利用公式（30.6）计算的模块 k 的测试工作量，式（30.7）的分母之和是系统所有模块的 Halstead 工作量的总和。

30.6.2 面向对象测试的度量

在 30.3 节提到的面向对象设计度量为设计质量提供了一种指标，它也为检查一个面向对象系统所需要的测试工作量提供了通用的指标。Binder[Bin94b] 提出了一组对面向对象系统

的“可测试性”具有直接影响的设计度量。该度量考虑了封装和继承方面。

方法缺少内聚 (Lack of Cohesion in Method, LCOM)[⊖]。LCOM 的值越高, 为保证方法不会产生副作用, 需要测试的状态越多。

公有与保护属性的百分比 (Percent Public and Protected, PAP)。公有属性是从其他类继承的, 因此对这些类是可见的。保护属性对子类的方法是可访问的。该度量指明类的公有属性或保护属性的百分比。PAP 的高值增加了类间副作用的可能性, 由于公有和保护属性导致较高的潜在耦合[⊖]。必须设计保证发现这些副作用的测试。

对数据成员的公有访问 (Public Access to Data Member, PAD)。这个度量指明可以访问另一个类属性的类 (或方法) 的数量, 这违背了封装。PAD 的高值导致类间的潜在副作用, 必须设计保证发现这些副作用的测试。

根类的数量 (Number of Root Class, NOR)。该度量是在设计模型中描述的不同类层次的计数。必须为每个根类和相应的类层次开发一组测试。当 NOR 增大时, 测试工作量也随之增加。

扇入 (Fan-IN, FIN)。当用于面向对象环境时, 在继承层次中的扇入是多继承的指示。FIN > 1 指示类从多个根类中继承属性和操作。应该尽可能避免 FIN > 1 的情况。

子女数 (Number of Children, NOC) 和继承树的深度 (Depth of the Inheritance Tree, DIT)[⊖]。如第 24 章所讨论的, 对每个子类, 必须重新测试超类的方法。

建议 面向对象测试是相当复杂的。基于测量特征, 度量可以帮助工程人员将测试资源锁定在值得“怀疑”的线程、场景和类的包。使用这些度量吧。

677

30.7 维护的度量

本章所介绍的所有软件度量均可用于新软件的开发和现有软件的维护。然而, 人们已提出了专门针对维护活动的度量。

IEEE 标准 982.1-2005[IEE05] 提出了一种软件成熟度指标 (Software Maturity Index, SMI), 它提供了对软件产品稳定性的指示 (基于产品每次发布所发生的变更)。可以确定以下信息:

M_T = 当前发布的模块数量

F_c = 当前发布中已变更的模块数量

F_a = 当前发布中已增加的模块数量

F_d = 当前发布中已删除前一发布中的模块数量

软件成熟度指标用下列方式计算:

$$SMI = \frac{M_T - (F_a + F_c + F_d)}{M_T}$$

当 SMI 的值接近 1.0 时, 产品开始稳定。SMI 也可用于软件维护活动计划的度量。产生软件产品的某个发布的平均时间可以与 SMI 联系起来, 且可以为维护工作量开发一个经验模型。

⊖ LCOM 的描述参见 30.3.3 节。

⊖ 一些人支持这样的观点: 设计中没有属性是共有的或私有的, 即 PAP=0。这意味着, 在其他类中的所有属性必须通过其方法来访问。

⊖ NOC 和 DIT 的描述参见 30.3.3 节。

软件工具 产品度量

[目标] 帮助软件工程师开发有意义的度量, 用以评估分析与设计建模、源代码生成以及测试期间所产生的工作产品的质量。

[机制] 这类工具涉及广泛的度量。它们要么作为独立的形式出现, 要么附在分析与设计、编码或测试工具中(这种比较常见)。在多数情况下, 度量工具分析软件的一种表示(例如, UML 模型或源代码), 并由此形成一种或多种度量。

[代表性工具]^①

- Krakatau Metrics。由 Power Software(www.powersoftware.com/products) 开发, 计算

C/C++ 和 Java 的复杂性度量、Halstead 度量及相关度量。

- Rational Rose。由 IBM (www-01.ibm.com/software/awdtools/developer/rose/) 发行, 它是一种综合的 UML 建模工具集, 并结合了很多度量分析特性。
- RSM。由 M-Squared Technologies(<http://msquaredtechnologies.com/resource-standard-metrics.html>) 开发, 它为 C、C++ 和 Java 计算各种面向代码的度量。
- Understand。由 Scientific Toolwork, Inc. (www.scitools.com) 开发, 它为各种编程语言计算面向代码的度量。

678

30.8 小结

软件度量为产品内部属性的质量评估提供了一种定量方法, 从而可以使软件工程师在产品开发出来之前进行质量评估。度量为创建有效的需求模型、设计模型、可靠的代码和严格的测试提供必要的理解。

为在现实世界中有用, 软件度量必须是简单的、可计算的、有说服力的、一致的和客观的。它应该与程序设计语言无关, 且能为软件工程师提供有效的反馈。

需求模型的度量侧重于需求模型的三个成分: 功能、数据和行为。设计度量考虑体系结构、构件级设计和界面设计问题。体系结构设计度量考虑设计模型的结构方面; 构件级设计度量通过为内聚性、耦合性和复杂性建立间接的测度, 提供了模块质量指示; 用户界面设计度量为 GUI 的易用性提供指标。WebApp 度量考虑了用户界面方面, 也考虑到了 WebApp 的美学、内容和导航。

面向对象系统的度量侧重于能应用于类与设计特征的测量: 局部化、封装、信息隐藏、继承及对象抽象技术。这些特征使类具有唯一性。CK 度量集定义了 6 个以类和类层次为重点的面向类的软件度量。该度量集也开发了度量, 用于评估类间协作以及类中方法的聚合度。在面向类级别, Lorenz 和 Kidd 提出的度量以及 MOOD 度量套件可以作为 CK 度量集的扩展。

Halstead 提供了一组令人感兴趣的源代码级度量。利用代码中出现的操作符和操作数的数量, 软件科学提供了多种度量, 用于评估程序质量。

很少有产品度量是直接针对软件测试和维护提出的。然而, 许多其他产品度量可用于指导测试过程, 且作为评估计算机程序可维护性的机制。人们已提出了大量的面向对象度量,

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

它们可用于评估面向对象系统的可测试性。

习题与思考题

- 30.1 测量理论是与软件度量密切相关的高级主题。利用 [Zus97]、[Fen91]、[Zus90] 或基于 Web 的资源，写一篇短文概括测量理论的主要原则。个人项目：关于这个主题准备一个演讲并在班上交流。
- 30.2 为什么不能为程序复杂性或程序质量开发一种单一的、全包容的度量？试着从日常生活中提出一些不符合 30.1.5 节定义的有效软件度量属性的测度或度量。
- 30.3 一个系统具有 12 个外部输入，24 个外部输出，30 个字段不同的外部查询，管理 4 个内部逻辑文件，与 6 个不同的遗留系统相连接（6 EIF）。所有这些数据属于平均复杂度，且整个系统相对简单。计算该系统的 FP。
- 30.4 软件系统 X 有 24 个功能需求和 14 个非功能需求。需求的确定性及完备性是什么？
- 30.5 某个主要的信息系统有 1140 个模块，其中 96 个模块完成控制和协调功能，490 个模块的功能依赖于前面的处理。该系统大约处理 220 个数据对象，每个对象平均有 3 个属性。存在 140 个唯一的数据库项和 90 个不同的数据库段。且 600 个模块有单一的入口和出口点。计算这个系统的 DSQI 值。
- 30.6 类 X 具有 12 个操作。在面向对象系统中，所有操作的环复杂度已计算出来，模块复杂度的平均值为 4。对于类 X ，操作 1 ~ 12 的复杂度分别为 5、4、3、3、6、8、2、2、5、5、4、4。计算每个类的加权方法度量。
- 30.7 开发一个软件工具，用以计算某编程语言模块的环复杂度，可以任选一种语言。
- 30.8 开发一个小型软件工具，用它对你选择的编程语言源代码进行 Halstead 分析。
- 30.9 一个遗留系统有 940 个模块，最新版本中需要变更其中的 90 个模块，此外，加入 40 个新模块，移除 12 个旧模块。计算这个系统的软件成熟度指标。

679

扩展阅读与信息资源

关于软件度量方面有大量的书籍，其中大部分都侧重于过程与项目度量，而不是产品度量。Jones 和 Bonsignour(《the Economic of Software Quality》，Addison-Wesley, 2011)、Lanza 和她的同事(《Object-Oriented Metrics in Practice》，Springer, 2010) 以及 Jones(《applied software measurements: global analysis of productivity and quality》，McGraw-Hill, 2008) 讨论了面向对象度量以及如何使用它们评估设计质量。Genero(《Metrics for Software Conceptual Models》，Imperial College Press, 2005) 和 Ejiogu(《Software Metrics》，BookSurge Publishing, 2005) 提出了很多关于用例、UML 模型和其他建模表示的技术度量。Hutcheson(《Software Testing Fundamentals: Methods and Metrics》，Wiley, 2003) 提出了一组测试的度量。Abran(《Software Metrics and Software Metrology, Wiley-IEEE computer society, 2010》)、Kan(《Metrics and Models in Software Quality Engineering》，Addison-Wesley, 2nd ed., 2002)、Fenton 和 Pfleeger(《Software Metrics: A Rigorous and Practical Approach》，Brooks-Cole Publishing, 1998) 以及 Zuse[Zus97] 对产品度量进行了全面的讨论。

Card 和 Glass[Car90]、Zuse[Zus90]、Fenton[Fen91]、Ejiogu[Eji91]、Moeller 和 Paulish(《Software Metrics》，Chapman and Hall, 1993) 以及 Hetzel[Het93] 都比较详细地讨论了产品度量。Oman 和 Pfleeger(《Applying Software Metrics》，IEEE Computer Society Press, 1997) 编辑出版了关于软件度量的重要论文集。

国际功能点用户组已经出版了一本关于使用功能度量的书籍(《The IFPUG Guide to OT and

Software Measurement, Auerbach Publications, 2012》)。Ebert 和他的同事(《Best Practices in Software Measurement》, Springer, 2004)讨论了建立度量程序的方法和软件测量的基本原则。Shepperd(《Foundations of Software Measurement》, Prentice-Hall, 1996)也详细地讨论了测量理论。当前的研究情况在《Proceedings of the Symposium on Software Metrics》(IEEE, 年刊)中有介绍。

[IEE93]全面总结了很多有用的软件度量。一般来讲,每个度量的讨论都对计算该度量所需的关键“基本方面”(测度)加以精炼,并总结了影响计算的适当关系,其附录中提供了讨论和许多参考文献。

Whitmire[Whi97]关于面向对象度量的讨论是迄今为止最全面和数学上最完善的。Lorenz 和 Kidd[Lor94]以及 Hendersen-Sellers(《Object-Oriented Metrics: Measures of Complexity》, Prentice-Hall, 1996)专门讨论了面向对象度量。

有关软件度量的大量信息资源可在网上获得。有关软件度量的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

680
682

管理软件项目

在本书的这一部分，将学习计划、组织和监控软件项目所需要的管理技术。在下面的章节中，我们将讨论以下问题：

- 在软件项目进行期间，为什么要对人员、过程和问题进行管理？
- 如何使用软件度量来管理软件项目和软件过程？
- 软件团队如何可靠地估算软件项目的工作量、成本和工期？
- 采用什么技术来正式评估影响项目成功的风险？
- 软件项目经理如何选择软件工作任务集？
- 如何编制项目进度计划？
- 为什么维护和再工程对于软件工程管理人员和实践人员都如此重要？

回答了这些问题后，就为管理软件项目做好了较充分的准备，便可以在某种程度上按时交付高质量的产品。

项目管理概念

要点浏览

概念: 虽然很多人在悲观的时候接受 Dilbert 的“管理”观点,但是在构建基于计算机的系统和产品时管理仍然是一项非常必要的活动。在软件从初始概念演化为可供有效使用的产品的过程中,项目管理涉及对人员、过程和所发生事件的计划和监控。

人员: 在软件项目中,每个人或多或少都做着“管理”工作。但是,管理活动的范围各不相同。软件工程师管理他们的日常活动,计划和监控技术任务。项目经理计划和监控软件工程师团队的工作。高级管理者协调业务和软件专业人员之间的关系。

重要性: 构建计算机软件是一项复杂的任务,尤其是当它涉及很多人员长期共同工作的时候。这就是为什么软件项目需要管理的原因。

步骤: 理解 4P——人员 (People)、产

品 (Product)、过程 (Process) 和项目 (Project)。必须将人员组织起来以有效地完成软件工作。必须和客户及其他利益相关者很好地沟通,以便了解产品的范围和需求;必须选择适合于人员和产品的过程;必须估算完成工作任务的工作量和工作时间,从而制定项目计划,包括定义工作产品、建立质量检查点以及确定一些机制来监控计划所规定的工作。

工作产品: 在管理活动开始时,首先是制定项目计划。该计划定义将要进行的过程和任务,安排工作人员,确定评估风险、控制变更和评价质量的机制。

质量保证措施: 在按时并在预算内交付高质量的产品之前,你不可能完全肯定项目计划是正确的。不过,作为项目经理,鼓励软件人员协同工作以形成一支高效的团队,并将他们的注意力集中到客户需求和产品质量上,这肯定是正确的。

Meiler Page-Jones [Pag85] 在其关于软件项目管理论著的序言中给出了以下一段陈述,这引起了许多软件工程顾问的共鸣:

我拜访了很多商业公司——好的和不好的,我又观察了很多 (IT) 管理者的业绩——好的和不好的。我常常恐惧地看到,这些管理者徒劳地与恶梦般的项目斗争着,在根本不可能完成的最后期限的压力下苦苦挣扎,或者是在交付了用户极为不满意的系统之后,又继续花费大量的时间去维护它。

Page-Jones 所描述的正是源于一系列管理和技术问题的症状。不过,如果在事后再剖析一下每个项目,很有可能发现一个共同的问题:项目管理太弱。

在本章以及第 32 ~ 37 章中,将给出进行有效的软件项目管理的关键

关键概念

敏捷团队
协调和沟通
关键实践
人员
问题分解
产品
项目
软件范围
软件团队
利益相关者
团队负责人
W²HH 原则

概念。本章考虑软件项目管理的基本概念和原则。第32章介绍过程和项目度量，这是做出有效管理决策的基础。第33章讨论用于估算成本的技术。第34章将帮助你编制实际的项目进度表。第35章阐述了进行有效的风险监测、风险缓解和风险控制的管理活动。第36章将考虑维护和再工程，并讨论处理遗留系统时将遇到的管理问题。最后，第37章讨论了研究和改进团队的软件工程过程的技术。

31.1 管理涉及的范围

有效的软件项目管理集中于4P，即人员、产品、过程和项目，它们的顺序不是任意的。任何管理者如果忘记了软件工程工作是人的智力密集的劳动，他就永远不可能在项目管理上取得成功；任何管理者如果在项目开发早期没有鼓励利益相关者之间的广泛交流，他就冒着为错误的问题构建了“良好的”解决方案的风险；对过程不在意的管理者可能冒着把有效的技术方法和工具插入真空中的风险；没有建立可靠的项目计划就开始工作的管理者将危及产品的成功。

31.1.1 人员

从20世纪60年代起人们就一直在讨论要培养有创造力、高技术水平的软件人员。实际上，“人的因素”的确非常重要，美国卡内基·梅隆大学的软件工程研究所（SEI）认识到这一事实——“每个组织都需要不断地提高他们的能力来吸引、发展、激励、组织和留住那些为实现其战略业务目标所需的劳动力”[Cur01]，并开发了一个人员能力成熟度模型（People-CMM）。

人员能力成熟度模型中针对软件人员定义了以下关键实践域：人员配备、沟通与协调、工作环境、业绩管理、培训、报酬、能力素质分析与开发、个人事业发展、工作组发展以及团队精神或企业文化培养等。People-CMM成熟度达到较高水平的组织，更有可能实现有效的软件项目管理实践。

People-CMM与软件能力成熟度集成模型（第37章）相伴而生，后者可指导组织创建一个成熟的软件过程。与软件项目的人员管理及人员结构相关的问题将在本章后面的内容中考虑。

31.1.2 产品

在制定项目计划之前，应该首先确定产品的目标和范围，考虑可选的解决方案，识别技术和管理上的限制。如果没有这些信息，就不可能进行合理的（精确的）成本估算，也不可能进行有效的风险评估和适当的项目任务划分，更不可能制定可管理的项目进度计划来给出意义明确的项目进展标志。

作为软件开发者，必须与其他利益相关者一同定义产品的目标和范围。在很多情况下，这项活动是作为系统工程或业务过程工程的一部分开始的，并一直持续到作为软件需求工程的第一步（第8章）。确定产品的目标只是识别出产品的总体目标（从利益相关者的角度），而不用考虑如何实现这些目标。而确定产品的范围，是要标识出产品的主要数据、功能和行为特性，而且更为重要的是，应以量化的方式界定这些特性。

建议 坚持敏捷过程方法（第5章）的人指出敏捷过程比其他过程更简单，这可能是真的。但是敏捷过程仍然有一个过程，敏捷软件工程依然需要规则。

了解产品的目标和范围之后,就要开始考虑备选解决方案。虽然这一步并不讨论细节,但可以使管理者和参与开发的人员根据给定的约束条件选择“最好”的方案,约束条件包括产品交付期限、预算限制、可用人员、技术接口以及其他各种因素。

31.1.3 过程

软件过程(第3~5章)提供了一个框架,在该框架下可以制定软件开发的综合计划。一小部分框架活动适用于所有软件项目,不用考虑其规模和复杂性。多种不同的任务集合(每一种集合都由任务、里程碑、工作产品以及质量保证点组成)使得框架活动适合于不同软件项目的特性和项目团队的需求。最后是普适性活动——如软件质量保证、软件配置管理、测量,这些活动覆盖了过程模型。普适性活动独立于任何一个框架活动,且贯穿于整个过程之中。

31.1.4 项目

我们实施有计划的、可控制的软件项目的主要理由是:这是我们知道的管理复杂事物的唯一方法。然而,软件团队仍需要努力。在对1998~2004年的250个大型软件项目的一份研究中,Capers Jones[Jon04]发现,“大约有25个项目被认为是成功的,达到了他们的计划、成本和质量目标;大约有50个项目延迟或超期在35%以下;而大约有175个项目经历了严重的延迟和超期,或者没有完成就中途夭折了。”虽然现在软件项目的成功率可能已经有所提高,但项目的失败率仍然大大高于它的应有值^①。

为了避免项目失败,软件项目经理和开发产品的软件工程师必须避免一些常见的警告信号,了解实施成功的项目管理的关键因素,还要确定计划和监控项目的一目了然的方法。这些问题将在31.5节及以后的章节中讨论。

引述 一个项目如同一次公路旅行,有些项目是简单的、常规性的,就像在白天开车去购物。但是值得做的大多数项目就像在夜晚驾驶一辆卡车离开公路驶入大山中一样。

Cem Kaner,
James Bach,
Bret Pettichord

31.2 人员

人们开发计算机软件,并取得项目的成功,是由于他们受过良好的训练并得到了激励。我们所有人,从高级工程副总裁到基层开发人员,常常认为人员是不成问题的。虽然管理者常常表态说人员是最重要的,但有时他们言行并不一致。本节将分析参与软件过程的利益相关者,并研究组织人员的方式,以实现有效的软件工程。

31.2.1 利益相关者

参与软件过程(及每一个软件项目)的利益相关者可以分为以下5类:

- 高级管理者负责定义业务问题,这些问题往往对项目产生很大影响。
- 项目(技术)管理者必须计划、激励、组织和控制软件开发人员。
- 开发人员拥有开发产品或应用软件所需的技能。

① 看到这些统计数据,人们自然会问计算机的影响又为何持续呈指数增长。我们认为,部分原因是:相当数量的“失败”项目在刚开始时就是构想拙劣的,客户很快就失去了兴趣(因为他们所需要的并不像他们最初想象的那样重要),进而取消了这些项目。

- 客户阐明待开发软件的需求，包括关心项目成败的其他利益相关者。
- 最终用户是软件发布成为产品后直接与软件进行交互的人。

687

每个软件项目都有上述人员的参与^①。为了获得高效率，项目团队必须以能够最大限度地发挥每个人的技术和能力的方式进行组织，这是团队负责人的任务。

31.2.2 团队负责人

项目管理是人员密集型活动，因此，胜任开发的人却常常有可能是拙劣的团队负责人，他们完全不具备管理人员的技能。正如 Edgemon 所说：“很不幸但却经常是这样，人们似乎碰巧落在项目经理的位置上，也就意外地成为项目经理”。[Edg95]

在一本关于技术领导能力的优秀论著中，Jerry Weinberg[Wei86] 提出了领导能力的 MOI 模型（Motivation（激励）、Organization（组织）、Ideas or Innovation（思想或创新））。

激励：（通过“推”或“拉”）鼓励技术人员发挥其最大才能的一种能力。

组织：形成能够将最初概念转换成最终产品的现有过程（或创造新的过程）的能力。

思想或创新：即使必须在特定软件产品或应用系统的约束下工作，也能鼓励人们去创造并让人感到有创造性的一种能力。

Weinberg 提出，成功的项目负责人应采用一种解决问题的管理风格。也就是说，软件项目经理应该注重理解要解决的问题，把握住涌现的各种意见，同时让项目团队的每个人都知（通过言语，更重要的是通过行动）质量很重要，不能妥协。

关于一个具有实战能力的项目经理应该具有什么特点，另一种观点 [Edg95] 则强调了以下 4 种关键品质。

解决问题。具有实战能力的软件项目经理能够准确地诊断出最为密切相关的技术问题和组织问题；能够系统地制定解决方案，适当地激励其他开发人员来实现该方案；能够将在过去项目中学到的经验应用到新环境中；如果最初的解决方案没有结果，能够灵活地改变方向。

管理者的特性。优秀的项目经理必须能够掌管整个项目。必要的时候要有信心进行项目控制，同时还要允许优秀的技术人员按照他们的本意行事。

成就。为了优化项目团队的生产效率，一位称职的项目经理必须奖励那些工作积极主动并且做出成绩的人。必须通过自己的行为表明出现可控风险并不会受到惩罚。

影响和队伍建设。具有实战能力的项目经理必须能够“理解”人。他必须能理解语言和非语言的信号，并对发出这些信号的人的要求做出反应。项目经理必须能在高压力的环境下保持良好的控制能力。

提问 当我们选择软件项目的负责人时，我们在寻找什么？

引述 用最简单的话来说，负责人是这样的人，他知道自己想去哪里，并起身朝那里走。

John Erskine

688

31.2.3 软件团队

几乎可以说有多少开发软件的组织，就有多少种软件开发人员的组织结构。不管怎么说，组织结构不能轻易改变。至于组织改变所产生的实际的和行政上的影响，并不在软件项目经理的责任范围内。但是，对新的软

引述 并非每个小组都是团队，并非每个团队都是有效的。

Glenn Parker

① 开发 WebApp 或移动 App 时，在内容创作方面需要其他非技术人员参与。

件项目中所直接涉及的人员进行组织，则是项目经理的职责。

“最好的”团队结构取决于组织的管理风格、团队里的人员数目与技能水平，以及问题的总体难易程度。Mantei[Man81]提出了规划软件工程团队结构时应该考虑的7个项目因素：

(1) 待解决问题的难度；(2) 开发程序的规模，以代码行或者功能点来度量；(3) 团队成员需要共同工作的时间（团队生存期）；(4) 能够对问题做模块化划分的程度；(5) 待开发系统的质量要求和可靠性要求；(6) 交付日期的严格程度；(7) 项目所需要的友好交流的程度。

提问 选择软件团队的结构时，应该考虑什么因素？

Constantine[Con93]提出了软件工程团队的4种“组织范型”：

提问 确定软件团队的结构时，我们有哪些选择？

1. 封闭式范型。按照传统的权利层次来组织团队。当开发与过去已经做过的产品相似的软件时，这种团队十分有效。但在这种封闭式范型下难以进行创新性的工作。
2. 随机式范型。松散地组织团队，团队工作依赖于团队成员个人的主动性。当需要创新或技术上的突破时，按照这种随机式范型组织的团队很有优势。但当需要“有次序地执行”才能完成工作时，这种团队就会陷入困境。
3. 开放式范型。试图以一种既具有封闭式范型的控制性，又包含随机式范型的创新性的方式来组织团队。工作是大家相互协作完成的。良好的沟通和根据团队整体意见做出决策是开放式范型的特征。开放式范型的团队结构特别适合于解决复杂的问题，但可能不像其他类型的团队那么有效率。
4. 同步式范型。依赖于问题的自然划分，组织团队成员各自解决问题的一部分，他们之间没有什么主动的交流。

689

从历史的角度看，最早的软件团队组织是封闭式范型结构，最初称为主程序员团队。这种结构首先由 Harlan Mills 提出，并由 Baker[Bak72] 描述出来。Constantine[Con93] 提出的随机式范型是主程序员团队结构的一个变种，主张建立具有独立创新性的团队，其工作方式可恰当地称为创新的无政府状态。尽管自由的软件工作方式是有吸引力的，但在绩效良好的团队中必须将创新能力作为软件工程组织的中心目标。为了建成一支绩效良好的团队，团队成员必须相互信任，团队成员的技能分布必须适合于要解决的问题，并且如果保持团队的凝聚力，必须将坚持个人己见的人员排除于团队之外。

引述 如果你要做得较好，那就竞争。如果你要做得极好，那就合作。

作者不详

无论是什么类型的团队，每个项目经理的目标都是帮助建立一支有凝聚力的团队。DeMarco 和 Lister[DeM98] 在其论著《Peopleware》中讨论了这个问题：

在商业界，我们往往随便使用团队这个词。任何被分配在一起工作的一组人都可以称为“团队”。但很多这样的小组看起来并不像团队，它们没有统一的对成功的定义，没有任何鲜明的团队精神。它们所缺少的是——一种很珍贵的东西，我们称之为“凝聚力”。

提问 什么是“有凝聚力”的团队？

一个有凝聚力的团队是一组团结紧密的人，他们的整体力量大于个体力量的总和……

一旦团队开始具有凝聚力，成功的可能性就大大提高。这个团队可以变得不可阻挡，成为成功的象征……他们不需要按照传统的方式进行管理，也不需要去激励。他们已经有了动力。

DeMarco 和 Lister 认为，同一般的团队相比，有凝聚力的团队成员具有更高的生产率和

更大的动力。他们拥有统一的目标和共同的文化，而且在很多情况下，“精英意识”使得他们独一无二。

但是，并非所有的团队都具有凝聚力。事实上，很多团队都受害于 Jackman [Jac98] 称之为“团队毒性”的东西。她定义了 5 个“培育潜在含毒团队环境”的因素：（1）狂乱的工作氛围；（2）引起团队成员间产生摩擦的重大挫折；（3）“碎片式的或协调很差”的软件过程；（4）在软件团队中没有清晰的角色定义；（5）“接连不断地重蹈覆辙”。

为了避免狂乱的工作环境，项目经理应该确保团队可以获取完成工作所需的所有信息；而且，主要目标一旦确定下来，除非绝对必要，否则不应该修改。给予软件团队尽可能多的决策权，这样能使团队避免挫败。通过理解将要开发的产品和完成工作的人员，以及允许团队选择过程模型，可以避免选择不适当的软件过程（如不必要的或繁重的工作任务，或没有很好地选择工作产品）。团队本身应该建立自己的责任机制（技术评审[⊖]是实现此目标的极好方式），并规定一系列当团队成员未能完成任务时的纠正方法。最后，避免失败的关键是建立基于团队的信息反馈方法和解决问题的技术。

除了 Jackman 描述的 5 个毒素以外，团队成员的个性不同也给软件团队带来了一系列的问题。有些团队成员性格外向，有些团队成员性格内向。有些人依靠直觉收集信息，从分离的事实中提炼主要概念；有些人则是线性地处理信息，从提供的数据中收集和组织微小的细节。有些团队成员只有在给出逻辑的、有序的论据时，才能做出决策；有些团队成员则依靠直觉，喜欢根据“感觉”做出决策。有些人希望有详细的任务进度计划，使得他们能够按部就班地完成项目的各个部分；有些人则喜欢更自发的环境，在这种环境中，允许开放地争论问题。有些人工作刻苦，在最后期限前很长时间就完成了任务，从而避免了时间逼近所带来的压力；而有些人则经常为了时限的逼近而在最后一分钟加班冲刺。熟练的团队负责人一般都掌握了如何帮助不同个性的人协同工作的方法。如何对待个性不同的人心理学问题，这超出了本书研究的范围[⊖]。不过，重要的是要注意到，识别人员差异是建立有凝聚力的团队的第一步。

31.2.4 敏捷团队

很多软件组织倡导将敏捷软件开发（第 5 章）作为解决软件项目工作中诸多困扰的一剂良方。回顾一下，敏捷方法学倡导的是：通过尽早地逐步交付软件来使客户满意；组织小型的充满活力的项目团队；采用非正式的方法；交付最小的软件工程工作产品；以及总体开发简易性。

小型的充满活力的项目团队，也称为敏捷团队，这种团队采纳了很多成功的软件项目团队的特性（在上一节内容中谈到的），避免了很多产生问题的毒素。同时，敏捷方法学强调团队成员的个人能力与团队协作精神相结合，这是团队成功的关键因素。对此，Cockburn 和 Highsmith [Coc01a] 这样写道：

如果项目成员足够优秀，那么他们几乎可以采用任何一种过程来完成任务。如果项目成员不够优秀，那么没有任何一种过程可以弥补这个不足。“人员胜过过程”阐明的正是这样

提问 为什么团队没有凝聚力？

引述 只有做或不做，没有尝试。
Yoda
（《星球大战》）

690

691

⊖ 技术评审在第 20 章详细讨论。

⊖ 关于这些问题（当它们和软件项目团队相关时）的一个很好的介绍可在 [Fer98] 中找到。

的含义。然而，如果缺乏用户和主管人员的支持，也可以毁掉一个项目，即“政策胜过人员”。缺乏支持可以阻止最好的人员完成任务。

在软件项目中，为了充分发挥每个团队成员的能力，并培养有效的合作，敏捷团队是自组织的。自组织团队不必保持单一的团队结构，而是采用 31.2.3 节讨论的由 Constantine 提出的随机、开放、同步式的范型。

很多敏捷过程模型（如 Scrum）给予敏捷团队相当大的自主权来进行项目管理，可以因工作需要做出技术决定。将计划制定工作压缩到最低程度，并且允许团队自己选择适用的手段（例如，过程、方法和工具），只受业务需求和组织标准的限制。在项目进展过程中，自组织团队关注的是在特定的时间点使项目获益最大的个人能力。为了做到这一点，敏捷团队召开日常团队例会，对当天必须完成的工作进行协调和同步控制。

基于在团队例会中获取的信息，团队能使他们所采用的手段不断适应持续增加的工作。当每一天过去的时候，连续的自组织和协作使团队朝着软件逐步接近的完工的目标前进。

关键点 敏捷团队是自组织的团队，拥有制定计划和做技术决定的自主权。

31.2.5 协调和沟通问题

使软件项目陷入困境的原因很多。许多开发项目规模很大，导致复杂性高、混乱、难以协调团队成员间的关系。不确定性是经常存在的，它会引起困扰项目团队的一连串的变更。互操作性已经成为许多系统的关键特性。新的软件必须与已有的软件通信，并遵从系统或产品所施加的预定义约束。

引述 集体所有权只不过是如下观念的产物：产品属于（敏捷）团队，而不属于团队中的个人。

Jim Highsmith

现代软件的这些特征（规模、不确定性和互操作性）确实都存在。为了有效地处理这些问题，必须建立切实可行的方法来协调工作人员之间的关系。为了做到这一点，需要建立团队成员之间以及多个团队之间的正式和非正式的交流机制。正式的交流机制是通过“文字、各级会议及其他相对而言非交互的和非个人的交流渠道”[Kra95]来实现的。非正式的交流机制则更加个人化。软件团队的成员在遇到特殊情况时交流意见，出现问题时请求帮助，而且在日常工作中彼此之间互相影响。

692

SafeHome 团队结构

[场景] SafeHome 软件项目启动之前，Doug Miller 的办公室。

[人物] Doug Miller，SafeHome 软件工程技术团队经理；Vinod Raman，Jamie Lazar 及其他产品软件工程团队成员。

[对话]

Doug：你们看过市场销售部准备的有关 SafeHome 的基本信息了吗？

Vinod（一边看着同事，一边点头）：是的，但我们有很多问题。

Doug：过一会儿再讨论这些问题。我们先

来讨论一下应该如何组织一个团队，哪些人应该负责……

Jamie：Doug，我对敏捷方法非常感兴趣，我想我们应该是一个自组织的团队。

Vinod：我同意。给定一个我们都能胜任的严格的期限和某些不确定性（笑），看起来是一种正确的方式。

Doug：我赞同，但是你们知道如何操作吗？

Jamie（边笑边说，好像在背诵什么）：我们做出战术决定，确定由谁做、做什么、

什么时间做。但按时交付产品是我们的责任。

Vinod: 还有质量。

Doug: 很正确,但是记住还有约束。市场部决定要生产的软件的增量,当然这要征求我们的意见。

Jamie: 还有?

Doug: 还有,要使用 UML 作为我们的建

模方法。

Vinod: 但要保持无关的文档减到最少。

Doug: 那谁和我联络?

Jamie: 我们确定 Vinod 作为技术负责人,因为他的经验最丰富。因此, Vinod 是你的联络人,但你应该自由地与每个人交流。

Doug: 别担心,我会的。

31.3 产品

从软件项目一开始,软件项目经理就面临着进退两难的局面。需要定量地估算成本和有组织地计划项目的进展,但却没有可靠的信息可以使用。虽然对软件需求的详细分析可以提供估算所需的信息,但需求分析常常需要数周甚至数月的时间才能完成。更糟糕的是,需求可能是不固定的,随着项目的进展经常会发生变化。然而,计划总是“眼前”就需要的!

不管喜欢与否,从项目一开始,就要研究应该开发哪些产品以及要解决哪些问题。至少,我们要建立和界定产品的范围。

693

31.3.1 软件范围

软件项目管理的第一项活动是确定软件范围。软件范围是通过回答下列问题来定义的:

项目环境。要开发的软件如何适应于大型的系统、产品或业务环境,该环境下要施加什么约束?

信息目标。软件要产生哪些客户可见的数据对象作为输出?需要什么数据对象作为输入?

功能和性能。软件要执行什么功能才能将输入数据变换成输出数据?软件需要满足什么特殊的性能要求?

软件项目范围在管理层和技术层都必须是无歧义的和可理解的。对软件范围的描述必须是界定的。也就是说,要明确给出定量的数据(例如,并发用户数、邮件列表的长度、允许的最大响应时间);说明约束和限制(例如,产品的成本要求会限制内存的大小),并描述其他的调节因素(例如,期望的算法能被很好地理解,并采用 Java 实现)。

建议 如果你不能把握待开发软件的某个特征,就将该特征作为一个项目风险列出(第35章)。

31.3.2 问题分解

问题分解,有时称为问题划分或问题细化,它是软件需求分析(第8~11章)的核心活动。在确定软件范围的活动中,并不试图去完全分解问题,只是分解其中的两个主要方面:(1)必须交付的功能和内容(信息);(2)所使用的过程。

在面对复杂的问题时,人们常常采用分而治之的策略。简单地说,就是将一个复杂的问题划分成若干更易处理的小问题。这是项目计划开始时

建议 为了制定合理的项目计划,你必须对问题进行分解。问题分解可以使用一系列的功能或者用例来进行,对于敏捷开发来说,使用的是用户故事。

31.4.2 过程分解

软件团队在选择最适合项目的软件过程模型时，应该具有很大的灵活性。一旦选定了过程模型，项目团队可以根据需要灵活地确定过程模型中应包含的软件工程任务。较小的项目如果与以前开发过的项目相似，则可以采用线性顺序方法。如果时间要求太紧，不可能完成所有功能，这时增量策略可能是最好的。同样，如果项目具有其他特性（如需求的不确定性、突破性的新技术、难以相处的客户、明显的复用潜力等），可能就要选择其他过程模型^①。

关键点 过程框架建立了项目计划的纲要，并通过分配适合项目的一系列任务对其进行调整。

一旦选定了过程模型，就要根据所选的过程模型对过程框架做适应性修改。但在所有情况下，前面讨论过的通用框架活动都可以使用。它既适用于线性模型，也适用于迭代和增量模型、演化模型，甚至是并发模型或构件组装模型。过程框架是不变的，是软件组织进行所有工作的基础。

但实际的工作任务是不同的。当项目经理问：“我们如何完成这个框架活动”时，就意味着过程分解开始了。例如，一个小型的比较简单的项目在沟通活动中可能需要完成下列工作任务：

1. 列出需澄清的问题清单。
2. 与利益相关者会面商讨需澄清的问题。
3. 共同给出范围陈述。
4. 和所有相关人员一起评审范围陈述。
5. 根据需要修改范围陈述。

这些事件可能在不到 48 小时的时间内发生。这是一种过程分解方式，这种方式适用于小型的比较简单的项目。

现在，考虑一个更复杂的项目，它的范围更广，具有更重要的商业影响。这样一个项目在沟通中可能需要完成下列工作任务：

1. 评审客户需求。
2. 计划并安排与全体利益相关者召开正式的、有人主持的会议。
3. 研究如何说明推荐的解决方案和现有的方法。
4. 为正式会议准备一份“工作文档”和议程。
5. 召开会议。
6. 共同制定能够反映软件的数据、功能和行为特性的微型规格说明。或者，从用户的角度出发建立描述软件的用例。
7. 评审每一份微型规格说明或用例，确认其正确性、一致性和无歧义性。
8. 将这些微型规格说明组装起来形成一份范围文档。
9. 和所有相关人员一起评审范围文档或用例集。
10. 根据需要修改范围文档或用例。

两个项目都执行了我们称之为沟通的框架活动，但第一个项目团队的软件工作任务只是第二个项目团队的一半。

^① 回忆一下，项目特性对项目团队的结构也有相当大的影响，见 31.2.3 节。

31.5 项目

为了成功地管理软件项目，我们必须了解可能会出现什么毛病，以便避免这些问题。在一篇关于软件项目的优秀论文中，John Reel[REE99]定义了若干预示信息系统项目正处于危险状态的信号。有些时候，软件人员不理解客户的需要，这导致产品范围定义得很糟糕。在一些项目中，变更没有得到很好的管理。有时，所选的技术发生了变化，或者业务需求改变了，或者失去了赞助。管理者可能设定了不切实际的最后期限，或者最终用户抵制这个新系统。有些情况下，项目团队不具有所需的技能。最后还可能是，有些开发人员似乎从来没有从自己的错误中学习。

提问 什么信号表示软件项目正处于危险状态？

在讨论特别困难的软件项目时，疲惫不堪的从业人员常常提及“90-90 规则”：系统前面 90% 的任务会花费所分配总工作量和时间的 90%，系统最后 10% 的任务也会花费所分配总工作量和时间的 90% [Zah94]。导致该 90-90 规则的根源就在上面列出的“信号”中。

这太消极了！管理者如何避免上面提到的这些问题呢？Reel[Ree99]针对软件项目提出了以下易于理解的方法，共包含 5 部分。

1. 在正确的基础上开始工作。通过以下两点来实现：首先努力（非常努力）地正确理解要解决的问题，然后为每个参与项目的人员设置现实的目标和期望。这一点又通过组建合适的开发团队（31.2.3 节）并给予团队工作时所需的自由、权力和技术而得到加强。
2. 保持动力。很多项目的启动都有一个良好的开端，但是，后来慢慢地开始瓦解。为了维持动力，项目经理必须采取激励措施使人员变动量保持绝对最小，团队应该重视它完成的每项任务的质量，而高层管理应该尽可能不干涉团队的工作。^①
3. 跟踪进展。对于软件项目而言，当工作产品（如模型、源代码、测试用例集）正在产生或被认可（通过技术评审）时，跟踪项目进展要作为质量保证活动的一部分。此外，可以收集软件过程和项目测量（第 32 章）数据，然后对照软件开发组织的平均数据来评估项目的进展。
4. 做出英明的决策。总体上，项目经理和软件团队的决策应该“保持项目的简单性”。只要有可能，就使用商用成品软件或现有的软件构件或模式，可以采用标准方法时避免定制接口，识别并避免显而易见的风险，以及分配比你认为的时间更多的时间来完成复杂或有风险的任务（你需要每一分钟）。
5. 进行事后分析。建立统一的机制，从每个项目中获取可学习的经验。评估计划的进度和实际的进度，收集和分析软件项目度量数据，从团队成员和客户处获取反馈，并记录所有的发现。

[697]

31.6 W⁵HH 原则

Barry Boehm[Boe96]在其关于软件过程和项目的优秀论文中指出：“你需要一个组织原则，对它进行缩减来为简单的项目提供简单的（项目）计划。”Boehm 给出了一种方法，该方法描述项目的目标、里程碑、进度、责任、管理和技术方法以及需要的资源。他称之为

^① 这句话的意思是：将官僚主义减少到最低程度，取消无关的会议，不再强调教条地依附于过程和项目规则。团队应该是自组织的，拥有自治权。

W⁵HH 原则。这种方法通过提出一系列问题，来导出对关键项目特性以及项目计划的定义：

为什么 (Why) 要开发这个系统？所有利益相关者都应该了解软件工作的商业理由是否有效。该系统的商业目的值得花费这些人力、时间和金钱吗？

提问 我们如何定义关键的项目特性？

将要做什么 (What)？定义项目所需的任务集。

什么时候 (When) 做？团队制定项目进度，标识出何时开展项目任务以及何时到达里程碑。

某功能由谁 (Who) 负责？规定软件团队每个成员的角色和责任。

他们的机构组织位于何处 (Where)？并非所有角色和责任均属于软件团队，客户、用户和其他利益相关者也有责任。

如何 (How) 完成技术工作和管理工作？一旦确定了产品范围，就必须定义项目的管理策略和技术策略。

每种资源需要多少 (How much)？对这个问题，需要在对前面问题回答的基础上通过估算 (第 33 章) 而得到。

Boehm 的 W⁵HH 原则可适用于任何规模和复杂度的软件项目。给出的问题为你和你的团队提供了很好的计划大纲。

31.7 关键实践

Airlie Council^①提出了一组“基于性能管理的关键软件实践”。这些实践一直“被高度成功的软件项目和组织（它们的‘底线’性能大大优于产业界的平均水平）普遍采用，并被认为的确是关键的”[Air99]。

关键实践^②包括：基于度量的项目管理 (第 32 章)，成本及进度的经验估算 (第 33 和 34 章)，获得价值跟踪 (第 34 章)，根据质量目标跟踪缺陷 (第 19 ~ 21 章)，人员计划管理 (第 31.2 节)。每一项关键实践都贯穿于本书的第四部分。

软件工具 项目管理的软件工具

这里列出的都是通用工具，适用于大部分项目管理活动。而特定的项目管理工具，如计划工具、评估工具和风险分析工具，将在后续章节中讲述。

[代表性工具]^③

- Projectmanagement.com (<http://www.projectmanagement.com>) 已经为项目开发了一套实用的检查表。

projectmanagement.com) 已经为项目开发了一套实用的检查表。

- Ittoolkit.com(www.ittoolkit.com) “收集了很多计划指南、过程模板和精巧的工作表”，可以从光盘上获取。

① Airlie Council 由美国国防部组织的软件工程专家组组成，其目的是开发在软件项目管理和软件工程中最佳的实践指南。关于最佳实践的更多信息参见 http://www.swqual.com/e_newsletter.html。

② 这里只讨论与“项目完整性”有关的关键实践。

③ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

31.8 小结

软件项目管理是软件工程的普适性活动。它先于任何技术活动之前开始，且持续贯穿于整个计算机软件的建模、构建和部署之中。

4P——人员、产品、过程和项目，对软件项目管理具有重大的影响。必须将人员组织成有效率的团队，激励他们完成高质量的软件工作，并协调他们实现有效的沟通。产品需求必须在客户与开发者之间进行交流，划分（分解）成各个组成部分，并分配给软件团队。过程必须适合于人员和问题。选择通用过程框架，采用合适的软件工程范型，并挑选工作任务集合来完成项目的开发。最后，必须采用确保软件团队能够成功的方式来组织项目。

在所有软件项目中，最关键的因素是人员。可以按照多种不同的团队结构来组织软件工程师，从传统的控制层次到“开放式范型”团队。可以使用多种协调和沟通技术来支持团队的工作。一般而言，技术评审和非正式的人与人交流对开发者最有价值。

项目管理活动包括测量与度量、估算与进度安排、风险分析、跟踪和控制。这些主题将在以后几章中进一步探讨。

习题与思考题

- 31.1 基于本章给出的信息和自己的经验，列举出能够增强软件工程师能力的“10条戒律”。即列出10条指导原则，使得软件人员能够在工作中发挥其全部潜力。
- 31.2 SEI的人员能力成熟度模型（People-CMM）定义了培养优秀软件人员的“关键实践域”（KPA）。你的老师将为你指派一个关键实践域，请你对它进行分析和总结。
- 31.3 描述3种现实生活中的实际情况，其中客户和最终用户是相同的人。也描述3种他们是不同人的情况。
- 31.4 高级管理者所做的决策会对软件工程团队的效率产生重大影响。请列举5个实例来说明这一点。
- 31.5 温习 Weinberg 的书 [Wei86]，并写出一份2~3页的总结，说明在使用 MOI 模型时应该考虑的问题。
- 31.6 在一个信息系统组织中，你被指派为项目经理。你的工作是开发一个应用程序，该程序类似于你的团队已经做过的项目，只是规模更大而且更复杂。需求已经由用户写成文档。你会选择哪种团队结构？为什么？你会选择哪（些）种软件过程模型？为什么？
- 31.7 你被指派为一个小型软件产品公司的项目经理。你的工作是开发一个有突破性的产品，该产品结合了虚拟现实的硬件和高超的软件。家庭娱乐市场的竞争非常激烈，因而完成这项工作的压力很大。你会选择哪种团队结构？为什么？你会选择哪些软件过程模型？为什么？
- 31.8 你被指派为一个大型软件产品公司的项目经理。你的工作是管理该公司已被广泛使用的字处理软件的新版本的开发。由于竞争激烈，因此已经规定了紧迫的最后期限，并对外公布。你会选择哪种团队结构？为什么？你会选择哪些软件过程模型？为什么？
- 31.9 在一个为遗传工程领域服务的公司中，你被指派为软件项目经理。你的工作是管理一个软件新产品的开发，该产品能够加速基因分类的速度。这项工作是面向研究及开发的，但其目标是在下一年度内生产出产品。你会选择哪种团队结构？为什么？你会选择哪些软件过程模型？为什么？
- 31.10 要求开发一个小型应用软件，它的作用是分析一所大学开设的每一门课程，并输出课程的平均成绩（针对某个学期）。写出该问题的范围陈述。
- 31.11 给出 31.3.2 节中讨论的页面布局功能的第一级功能分解。

扩展阅读与信息资源

项目管理研究所 (PMI) 出版的书 (《 Guide to the Project Management Body of Knowledge 》, 4th ed., PMI, 2009) 介绍了项目管理的所有重要方面。Wysocki (《 Effective Software Project Management: Traditional, Agile, Extreme 》, 6th ed., Wiley, 2011)、Slinger 和 Broderick (《 The Software Project Manager's Bridge to Agility 》, Addison-Wesley, 2008)、Bechtold (《 Essentials of Software Project Management 》, 2nd ed., Management Concepts, 2007)、Stellman 和 Greene (《 Applied Software Project Management 》, O'Reilly, 2005) 以及 Berkun (《 Making Things Happen: Mastering Project Management Theory in Practice 》, O'Reilly, 2008) 讲授了基本技能, 并为所有的软件项目管理任务提供了详细的指南。McConnell (《 Professional Software Development 》, Addison-Wesley, 2004) 提供了实用的建议, 指导如何获得“更短的进度计划、更高质量的产品和更成功的项目”。Henry (《 Software Project Management 》, Addison-Wesley, 2003) 为所有项目经理提供了现实的建议。

Tom DeMarco 和他的同事 (《 Adrenaline Junkies and Template Zombies 》, Dorset House, 2008) 对每种软件项目中会遇到的人员模式都给出了有深入见解的处理方法。由 Weinberg 所著的一套 4 册的系列丛书 (《 Quality Software Management 》, Dorset House, 1992, 1993, 1994, 1996) 介绍了基本的系统思想和管理概念, 讲解了如何有效地使用测量, 并且说明了“一致的行为”, 即建立管理者需要、技术人员需要以及商业需要之间的协调关系。它不仅为新的管理者同时也为有经验的管理者提供了有用的信息。Futrell 和他的同事 (《 Quality Software Project Management 》, Prentice-Hall, 2002) 提出了大量的项目管理处理方法。Neill 和他的同事 (《 Antipatterns: Managing Software Organizations 》, 2nd ed., Auerbach Publications, 2011) 以及 Brown 和他的同事 (《 Antipatterns in Project Management 》, Wiley, 2000) 在其所著的书中讨论了软件项目管理期间不能做的事情。

Brooks (《 The Mythical Man-Month 》, Anniversary Edition, Addison-Wesley, 1995) 更新了他的杰作, 给出了关于软件项目及管理问题的新见解。McConnell (《 Software Project Survival Guide 》, Microsoft Press, 1997) 为那些必须管理软件项目的人提供了卓越的有实效的行动指南。Purba 和 Shah (《 How to Manage a Successful Software Project 》, 2nd ed., Wiley, 2000) 提供了很多案例研究, 指出为什么一些项目能成功, 而另外一些项目却会失败。Kerzner (《 Project Management: A Systems Approach to Planning Scheduling and Controlling 》, 10th ed., Wiley, 2009) 和 Bennatan (《 On Time Within Budget 》, 3rd ed., Wiley, 2000) 在其所著的书中为软件项目经理提供了实用的提示和指导。Weigers (《 Practical Project Initiation 》, Microsoft Press, 2007) 为软件项目的成功实施提供了实用的指南。

可以证明, 软件项目管理中最重要的是人员管理。Cockburn (《 Agile Software Development 》, Addison-Wesley, 2002) 给出了关于软件人员的论述, 这是到目前为止关于软件人员的最好的论述之一。DeMarco 和 Lister[DeM98] 撰写了关于软件人员和软件项目的最权威著作。此外, 关于这一主题, 近年来出版的如下书籍值得一读:

Cantor, M., 《 Software Leadership: A Guide to Successful Software Development 》, Addison-Wesley, 2001。

Carmel, E., 《 Global Software Teams: Collaborating Across Borders and Time Zones 》, Prentice Hall, 1999。

Chandler, H. M., 《 Game Production Handbook 》, 2nd ed., Charles River Media, 2008。

Constantine, L., 《 Peopleware Papers: Notes on the Human Side of Software 》, Prentice Hall, 2001。

Ebert, C., 《 Global Software and IT: A Guide to Distributed Development, Projects, and

Outsourcing》, Wiley-IEEE Computer Society, 2011。

Fairley, R. E., 《Managing and Leading Software Projects》, Wiley-IEEE Computer Society, 2009。

Garton, C., and Wegryn, K., 《Managing Without Walls》, McPress, 2006。

Humphrey, W. S., and Over, J. W., 《Leadership, Teamwork, and Trust: Building a Competitive Software Capability》, Addison-Wesley, 2011。

Humphrey, W. S., 《Managing Technical People: Innovation, Teamwork, and the Software Process》, Addison-Wesley, 1997。

Humphrey, W. S., 《TSP-Coaching Development Teams》, Addison-Wesley, 2006。

Jones, P. H., 《Handbook of Team Design: A Practitioner's Guide to Team Systems Development》, McGraw-Hill, 1997。

Karolak, D. S., 《Global Software Development: Managing Virtual Teams and Environments》, IEEE Computer Society, 1998。

Misrik, I., et al., 《Collaborative Software Engineering》, Springer, 2010。

Peters, L., 《Getting Results from Software Development Teams》, Microsoft Press, 2008。

Whitehead, R., 《Leading a Software Development Team》, Addison-Wesley, 2001。

以下列出的一些畅销的“管理”书籍并不和软件世界特别相关, 有的还过于简单、过于概括: Kanter (《Confidence》, Three Rivers Press, 2006), Covey (《The 8th Habit》, Free Press, 2004), Bossidy (《Execution: The Discipline of Getting Things Done》, Crown Publishing, 2002), Drucker (《Management Challenges for the 21st century》, Harper Business, 1999), Buckingham 和 Coffman (《First, Break All the Rules: What the World's Greatest Managers Do Differently》, Simon and Schuster, 1999), 以及 Christensen (《The Innovator's Dilemma》, Harvard Business School Press, 1997)。这些书强调由快速变化的经济定义的“新规则”。比较老的书如《Who Moved My Cheese?》《The One-Minute Manager》和《In Search of Excellence》, 它们仍然很有价值, 能够帮助你更有效地管理人员和项目。

在网上可以获得大量的关于软件项目管理的信息。最新的参考文献可在 SEPA 网站 www.mhhe.com/pressman 下的“software engineering resources”中找到。

过程度量与项目度量

要点浏览

概念: 软件过程度量和项目度量是定量的测量, 这些测量能使你更深入地了解软件过程的功效, 以及使用该过程作为框架进行开发的项目的功效。做度量时, 首先要收集基本的质量数据和生产率数据, 然后分析这些数据、与过去的平均值进行比较, 通过评估来确定是否有质量的改进和生产率的提高。度量也可以用来查明问题区域, 确定恰当的补救措施, 从而改进软件过程。

人员: 软件度量由软件管理者来分析和评估。测量数据通常由软件工程师来收集。

重要性: 如果不进行测量, 就只能根据主观评价来做判断。通过测量, 可以发现趋势(好的或坏的), 可以更好地进行估

算, 并且随着时间的推移, 软件能够获得真正的改进。

步骤: 首先确定一组数量有限的易于收集的过程测量、项目测量和产品测量。通常使用面向规模或面向功能的度量对这些测量进行规范化。然后对测量结果进行分析, 与该组织以前完成的类似项目的平均数据进行比较。最后评估趋势, 给出结论。

工作产品: 一组软件度量, 它们提供了对过程深入透彻的认识和对项目的理解。

质量保证措施: 通过采用一致而简单的测量计划来保障, 但该计划绝对不能用于对个人表现的评估、奖励或惩罚。

通过提供目标评估的机制, 测量能使我们对过程和项目有更深入的了解。Lord Kelvin 曾经说过:

当你能够测量你所说的事物, 并能用数字表达时, 你就对它有了一定的了解; 如果不能测量它, 也不能用数字表达时, 就说明你对它的了解还很贫乏, 不能令人满意。后者可能是知识的起点, 但你在思想上还远远没有达到科学的境地。

软件工程界已经认可了 Lord Kelvin 的话。但这并不是一帆风顺, 也不是只有一点点争论。

将测量应用于软件过程, 目的是持续改进软件过程。也可以将测量应用于整个软件项目, 辅助进行估算、质量控制、生产率评估及项目控制。软件工程师可以使用测量来帮助评估工作产品的质量, 在项目进展过程中辅助进行战术决策(第 30 章)。

从软件过程以及使用该过程进行开发的项目出发, 软件团队主要关注生产率度量和质量度量——前者是对软件开发“输出”的测量, 它是投入的工作量和时间的函数, 后者是对所生产的工作产品“适用性”的测量。为了进行计划和估算, 需要了解历史数据。以往项目的

软件开发生产率是多少？开发出来的软件质量怎么样？怎样利用以往的生产率数据和质量数据推断现在的生产率和质量？这些数据如何帮助我们更精确地计划和估算？

在 Park、Goethert 和 Florac[Par96b] 的关于软件测量的指导手册中，他们讨论了进行测量的理由：（1）通过刻画而“获得对过程、产品、资源和环境的了解，建立同未来评估进行比较的基线”；（2）通过评价来确定“相对于计划的状况”；（3）“通过理解过程和产品间的关系，并构建这些关系的模型来进行预测”；（4）“通过识别难点、根本原因、低效率和其他提高产品质量和过程性能的机会来进行改进”。

测量是一个管理工具，如果能正确地使用，它将为项目管理者提供洞察力。因此，测量能够帮助项目管理者 and 软件团队制定出使项目成功的决策。

32.1 过程领域和项目领域中的度量

过程度量的收集涉及所有的项目，要经历相当长的时间，目的是提供能够引导长期的软件过程改进（第 37 章）的一组过程指标。项目度量使得软件项目管理者能够：（1）评估正在进行中的项目的状态；（2）跟踪潜在的风险；（3）在问题造成不良影响之前发现它们；（4）调整工作流程或任务；（5）评估项目团队控制软件工作产品质量的能力。

测量数据由项目团队收集，然后被转换成度量数据在项目期间使用。测量数据也可以传送给那些负责软件过程改进的人员。因此，很多相同的度量既可用于过程领域，又可用于项目领域。

32.1.1 过程度量和软件过程改进

改进任何过程的唯一合理方法就是测量该过程的特定属性，再根据这些属性建立一组有意义的度量，然后使用这组度量提供的指标来导出过程改进策略（第 37 章）。但是在讨论软件度量及其对软件过程改进的影响之前，必须注意到：过程仅是众多“改进软件质量和组织性能的控制因素”中的一种 [Pau94]。

关键概念

- 缺陷排除效率 (DRE)
- 功能点 (FP)
- 测量
- 度量
- 争论
- 基线
- 制定大纲
- 面向功能
- 基于 LOC 的度量
- 面向对象
- 私有和公有
- 过程
- 生产率
- 项目
- 面向规模
- 软件质量
- 面向用例
- WebAPP

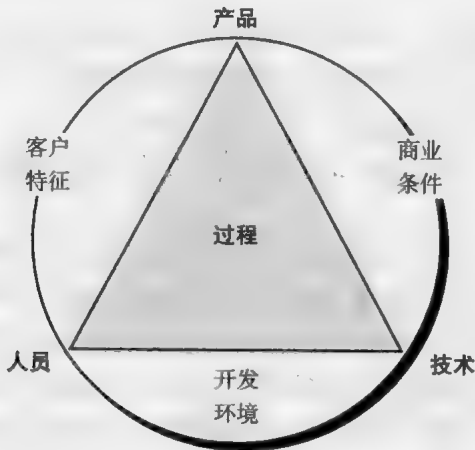


图 32-1 软件质量和组织有效性的决定因素 [Pau94]

704

在图 32-1 中,过程位于三角形的中央,连接了三个对软件质量和组织绩效有重大影响的因素。其中,人员的技能和动力 [Boe81] 被认为是对质量和绩效影响最大的因素,产品复杂性对质量和团队绩效也有相当大的影响,过程中采用的技术(即软件工程方法和工具)也有一定的影响。

另外,过程三角形位于环境条件圆圈内,环境条件包括:开发环境(如集成的软件工具)、商业条件(如交付期限、业务规则)、客户特征(如交流和协作的难易程度)。

软件过程的功效只能间接地测量。也就是说,根据从过程中获得的结果来导出一组度量。这些结果包括:在软件发布之前发现的错误数的测度,提交给最终用户并由最终用户报告的缺陷的测度,交付的工作产品(生产率)的测度,花费的工作量的测度,花费时间的测度,与进度计划是否一致的测度,以及其他测度。也可以通过测量特定软件工程任务的特性来导出过程度量。例如,测量第 2 章中所描述的普适性活动和一般软件工程活动所花费的工作量和时间。

Grady [Gra92] 认为不同类型过程数据的使用可以分为“私有的和公有的”。软件工程师可能对他个人收集的度量的使用比较敏感,这是很自然的事。这些数据对本人应该是私有的,是仅供本人参考的指标。私有度量的例子有:缺陷率(个人)、缺陷率(软件构件)和开发过程中发现的错误数。

“私有过程数据”的观点与 Humphrey [Hum05] 所提出的个人软件过程方法(第 4 章)相一致。Humphrey 认为软件过程改进能够也应该开始于个人级。私有过程数据是改进自身软件工程方法的重要驱动力。

有些过程度量是软件项目团队私有的,但对团队所有成员是公有的。例如,主要软件功能(由多个开发人员共同完成)的缺陷报告、技术评审发现的错误,以及每个构件或功能的代码行数或功能点数^①。团队评审这些数据,以找出能够提高团队效能的指标。

公有度量一般吸收了原本属于个人或团队的私有信息。收集和评估项目级的缺陷率(绝对不能归咎于某个人)、工作量、时间以及相关的数据,来找出能够改善组织的过程性能的指标。

软件过程度量对于组织提高其过程成熟度的整体水平能够提供很大的帮助。不过,与其他所有度量一样,软件过程度量也可能被误用,产生的问题比它们所能解决的问题更多。Grady [Gra92] 提出了一组“软件度量规则”。管理者和开发者在制定过程度量大纲时,这些规则都适用:

- 解释度量数据时使用常识,并考虑组织的敏感性。
- 向收集测量和度量的个人及团队定期提供反馈。
- 不要使用度量去评价个人。
- 与开发者和团队一起设定清晰的目标,并确定为达到这些目标需要使用的度量。
- 不要用度量去威胁个人或团队。

关键点 软件人员的技能和动力是影响软件质量的最重要因素。

引述 软件度量让你知道什么时候笑,什么时候哭。

Tom Gilb

提问 对软件度量的私有使用和公有使用有什么不同?

提问 当我们收集软件度量时,应该采用什么指导原则?

705

① 代码行和功能点度量将在第 32.2.1 和 32.2.2 节中讨论。

- 指出问题区域的度量数据不应该被“消极地”看待，这些数据仅仅是过程改进的指标。

- 不要在某一个别的度量上纠缠，而无暇顾及其他重要的度量。

随着一个组织更加得心应手地收集和使用过程度量，简单的指标获取方式会逐渐被更精确的称为统计软件过程改进（statistical software process improvement, SSPI）的方法所取代。本质上，SSPI 使用软件失效分析方法来收集在应用软件、系统或产品的开发及使用过程中所遇到的所有错误及缺陷信息^①。

32.1.2 项目度量

软件过程度量用于战略目的，而软件项目测量则用于战术目的。也就是说，项目管理者与软件项目团队通过使用项目度量及从中导出的指标，可以改进项目工作流程和技术活动。

在大多数软件项目中，项目度量的第一次应用是在估算阶段。那些从以往项目中收集的度量可以作为当前软件工作的工作量及时间估算的基础。随着项目的进展，将所花费的工作量及时间的测量与最初的估算值（及项目进度）进行比较。项目管理者可以使用这些数据来监控项目的进展。

随着技术工作的启动，其他项目度量也开始有意义了。生产率可以根据创建的模型、评审时间、功能点以及交付的源代码行数来测量。此外，对每个软件工程任务中发现的错误也要进行跟踪。在软件从需求到设计的演化过程中，需要收集技术度量（第 30 章）来评估设计质量，并提供若干指标，这些指标将会影响代码生成及测试所采用的方法。

项目度量的目的是双重的。首先，利用度量能够对开发进度进行必要的调整，以避免延迟，并减少潜在的问题和风险，从而使开发时间减到最短。其次，项目度量可用于在项目进行过程中评估产品质量，必要时可调整技术方法以提高质量。

提问 在项目中，我们应该如何使用度量？

随着质量的提高，缺陷会越来越少。随着缺陷数的减少，项目所需的修改工作量也会减少，从而降低项目的总体成本。

SafeHome

建立度量方法

[场景] SafeHome 软件项目即将启动，在 Doug Miller 的办公室。

[人物] Doug Miller, SafeHome 软件团队经理；Vinod Raman 和 Jamie Lazar, 产品软件工程团队成员。

[对话]

Doug：在项目工作开始之前，我想你们应该定义并收集一组简单的度量。首先，必须确定目标。

Vinod（皱着眉头）：以前，我们从来没有

做过这些，并且……

Jamie（打断他的话）：基于时间线的管理已经讨论过了，我们根本没有时间。度量到底有什么好处？

Doug（举手示意停止发言）：大家且慢，停一下。正因为我们以前从来没有做过度量，所以现在更要开始做。并且我说的度量工作根本不会占用很多时间，事实上，它只会节省我们的时间。

Vinod：为什么？

① 在本书中，错误（error）是指软件工程工作产品中的瑕疵（flaw），这些瑕疵在交付给最终用户之前已经被发现。而缺陷（defect）是指交付给最终用户之后才发现的瑕疵。应该注意到，其他人并没有进行这样的区分。

Doug：你看，随着我们的产品更加智能化，变得支持 Web、移动端等，我们将要做更多的内部软件工程工作。我们需要了解软件开发的过程，并改进过程，使我们能够更好地开发软件。要实现这一点，唯一的方法就是测量。

Jamie：但是我们的时间很紧迫，Doug。我不赞同太多琐碎的文字工作，我们需要时间来完成工作，而不是收集数据。

Doug：Jamie，工程师的工作包括收集数据、评估数据、使用评估结果来改进产品和过程。我错了吗？

Jamie：不，但是……

Doug：如果我们限定要收集的测量数不超过 5 个或 6 个，并集中关注质量方面，那

会怎么样？

Vinod：没有人能够反对高质量……

Jamie：对，但是，我不知道，我仍然认为这是不必要的。

Doug：在这个问题上，请听我的！关于软件度量你们了解多少？

Jamie（看着 Vinod）：不多。

Doug：这是一些 Web 参考资料，花几个小时读完。

Jamie（微笑着）：我还认为你说的这件事不会花费任何时间。

Doug：花费在学习上的时间绝对不会浪费，去做吧！然后我们要建立一些目标，提几个问题，定义我们需要收集的度量。

32.2 软件测量

第 30 章谈到物质世界中的测量可以分为两种方法：直接测量（如螺栓的长度）和间接测量（如螺栓的“质量”，通过统计废品数量来测量）。软件度量也可以这样来划分。

软件过程的直接测量包括花费的成本和工作量。产品的直接测量包括产生的代码行（LOC）、运行速度、存储容量以及某段时间内报告的缺陷；产品的间接测量包括功能、质量、复杂性、效率、可靠性、可维护性，以及许多在第 19 章中谈到的其他“产品特性”。

构造软件所需的成本和工作量、产生的代码行数以及其他直接测量都是相对容易收集的，只要事先建立特定的测量协议即可。但是，软件的质量和功能、效率或可维护性则很难获得，只能间接地测量。

我们已将软件度量范围分为过程度量、项目度量和产品度量。注意，产品度量对个人来讲是私有的，常常将它们合并起来生成项目度量，而项目度量对软件团队来说是公有的。再将项目度量联合起来可以得到整个软件组织公有的过程度量。但是，一个组织如何将来自不同个人或项目的度量结合起来呢？

为了说明这个问题，看一个简单的例子。两个不同项目团队中的人将他们在软件过程中发现的所有错误进行了记录和分类。然后，将这些个人的测量结合起来就产生了团队的测量。在软件发布前，团队 A 在软件过程中发现了 342 个错误，团队 B 发现了 184 个错误。所有其他情况都相同，那么在整个过程中哪个团队能更有效地发现错误呢？由于不了解项目的规模或复杂性，所以不能回答这个问题。不过，如果度量采用规范化的方法，就有可能产生在更大的组织范围内进行比较的软件度量。

引述 并非能够被计算的每件事物都有价值，也并非没有价值的每件事物都能够被计算。

Albert Einstein

708

建议 因为有很多因素会影响软件工作，因此不要用度量去比较个人或团队。

32.2.1 面向规模的度量

面向规模的软件度量是通过对质量和生产率的测量进行规范化后得到的，而这些测量都是根据开发过的软件的规模得到的。如果软件组织一直在做简单记录，就会产生一个如图 32-2 所示的面向规模测量的表。该表列出了在过去几年中完成的每一个软件开发项目及其相关的测量数据。查看 alpha 项目的数据（图 32-2）：花费了 24 人月的工作量，成本为 168000 美元，产生了 12100 行代码。需要提醒大家的是，表中记录的工作量和成本涵盖了所有软件工程活动（分析、设计、编码及测试），而不仅仅是编码。有关 alpha 项目更进一步的信息包括：产生了 365 页文档，在软件发布之前发现了 134 个错误，在软件发布给客户之后运行的第一年中遇到了 29 个缺陷，有 3 个人参加了 alpha 项目的软件开发工作。

709

项目	代码行	工作量	成本 (千美元)	文档页数	错误	缺陷	人员
alpha	12100	24	168	365	134	29	3
beta	27200	62	440	1224	321	86	5
gamma	20200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

图 32-2 面向规模的度量

为了得到能和其他项目同类度量进行比较的度量，你可以选择代码行作为规范化值。根据表中包含的基本数据，每个项目都能得到一组简单的面向规模的度量：

- 每千行代码（KLOC）的错误数
 - 每千行代码（KLOC）的缺陷数
 - 每千行代码（KLOC）的成本
 - 每千行代码（KLOC）的文档页数
- 此外，还能计算出其他有意义的度量：

- 每人月错误数
- 每人月千行代码数
- 每页文档的成本

面向规模的度量是否是软件过程测量的最好方法，对此并没有普遍一致的观点。大多数争议都围绕着使用代码行（LOC）作为关键的测量是否合适。LOC 测量的支持者声称：LOC 是所有软件开发项目的“产物”，并且很容易进行计算；许多现有的软件估算模型都是使用 LOC 或 KLOC 作为关键的输入；而且已经有大量的文献和数据都涉及 LOC。另一方面，反对者则认为，LOC 测量依赖于程序设计语言；当考虑生产率时，这种测量对设计得很好但较短的程序会产生不利的评价；它们不适用于非过程语言；而且在估算时需

关键点 面向规模的度量已经得到了广泛的应用，但对其有效性和适用性的争论一直在持续。

要一些可能难以得到的信息（例如，计划人员必须在分析和设计远未完成之前，就要估算出将产生的 LOC）。

32.2.2 面向功能的度量

面向功能的软件度量以功能（由应用程序提供）测量数据作为规范化值。应用最广泛的面向功能的度量是功能点（Function Point, FP）。功能点是根据软件信息域的特性及复杂性来计算的，它的计算方法已经在第 30 章中讨论过了[⊖]。

710

与 LOC 测量一样，功能点测量也是有争议的。支持者认为 FP 与程序设计语言无关，对于使用传统语言和非过程语言的应用系统来说，它都是比较理想的，而且它所依据的数据是在项目开发初期就可能得到的数据。因此，FP 是一种更有吸引力的估算方法。反对者则声称这种方法需要某种“熟练手法”，因为计算的依据是主观的而非客观的数据，信息域（及其他方面）的数据可能难以在事后收集。而且，FP 没有直接的物理意义，它仅仅是一个数字而已。

32.2.3 调和代码行度量和功能点度量

代码行和功能点之间的关系依赖于实现软件所采用的程序设计语言及设计的质量。很多研究试图将 FP 测量和 LOC 测量关联起来。表 32-1[⊖] [QSM02] 给出了在不同的程序设计语言中实现一个功能点所需的平均代码行数的粗略估算。

表 32-1 各种程序设计语言实现一个功能点所需的代码行数的粗略估算

程序设计语言	LOC/FP			
	平均值	中值	低值	高值
Ada	154	—	104	205
ASP	56	50	32	106
Assembler	337	315	91	694
C	148	107	22	704
C++	59	53	20	178
C#	58	59	51	704
COBOL	80	78	8	400
ColdFusion	68	56	52	105
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Focus	43	42	32	56
FORTRAN	90	118	35	—
FoxPro	32	35	25	35
HTML	43	42	35	53
Informix	42	31	24	57
J2EE	57	50	50	67
Java	55	53	9	214

⊖ 对 FP 计算的详细讨论参见 30.2.1 节。
⊖ 表中列出的数据是 Quantitative Software Management (www.qsm.com) 所开发数据的缩略版，已得到他们的使用许可，copyright 2002。

(续)

程序设计语言	LOC/FP			
	平均值	中值	低值	高值
JavaScript	54	55	45	63
JSP	59	—	—	—
Lotus Notes	23	21	15	46
Mantis	71	27	22	250
Natural	51	53	34	60
.NET	60	60	60	60
Oracle	42	29	12	217
OracleDev2K	35	30	23	100
PeopleSoft	37	32	34	40
Perl	57	57	45	60
PL/1	58	57	27	92
Powerbuilder	28	22	8	105
RPG II/III	61	49	24	155
SAS	50	35	33	49
Smalltalk	26	19	10	55
SQL	31	37	13	80
VBScript	38	37	29	50
Visual Basic	50	52	14	276

由这些数据可以看出，C++ 的一个 LOC 所提供的“功能”大约是 C 的一个 LOC 所提供功能的 2.4 倍（平均来讲）。Smalltalk 一个 LOC 所提供的“功能”至少是传统程序设计语言（如 Ada、COBOL 或 C）的 4 倍。利用上表包含的信息，只要知道了程序设计语言的语句行数，就可以“逆向”[Jon98] 估算出现有软件的功能点数量。

LOC 和 FP 测量经常用来导出生产率度量，这总会引起关于这些数据使用的争论。一个小组的 LOC/人月（或功能点/人月）应该与另一个小组的类似数据进行比较吗？管理者应该使用这些度量来评价个人绩效吗？对这些问题都斩钉截铁地回答“不！”原因是生产率受很多因素的影响，进行“苹果和橘子”式的比较很容易产生曲解。

人们发现，基于功能点的度量和基于 LOC 的度量都是对软件开发工作量和成本的比较精确的判定。为了使用 LOC 和 FP 进行估算（第 33 章），还必须建立一个历史信息基线。

在过程度量和项目度量中，主要应该关心生产率和质量——软件开发“输出量”（作为投入的工作量和时间的函数）的测量以及对生产的工作产品“适用性”的测量。为了进行过程改进和项目计划，必须掌握历史情况。在以往的项目中，软件开发的生产率是多少？生产的软件质量如何？怎样利用以往的生产率数据和质量数据推断现在的生产率和质量？如何利用这些数据帮助我们改进过程，以及更精确地规划新的项目？

32.2.4 面向对象的度量

传统的软件项目度量（LOC 或 FP）也可以用于估算面向对象的软件项目。但是，这些度量并没有提供对进度和工作量进行调整的足够的粒度，而这却是在演化模型或增量模型中进行迭代时所必需的。Lorenz 和 Kidd[Lor94] 提出了下列用于 OO 项目的度量。

场景脚本的数量。场景脚本（类似于用例）是一个详细的步骤序列，用来描述用户和应用之间的交互。每个脚本采用如下三个一组的形式来组织：

{ 发起者，动作，参与者 }

其中，发起者是指请求某个服务（首先传递一个消息）的对象，动作是该请求的结果，参与者是满足该请求的服务对象。场景脚本的数量与应用的规模及测试用例（一旦构建出系统，就必须设计测试用例来测试该系统）的数量紧密相关。

关键类的数量。关键类是“高度独立的构件”[Lor94]，在面向对象分析的早期进行定义（第 10 章）^①。由于关键类是问题域的核心，因此，这些类的数量既是开发软件所需工作量的指标，也是系统开发中潜在的复用数量的指标。

支持类的数量。支持类是实现系统所必需的但又不与问题域直接相关的类。例如，用户界面（UI）类、数据库访问及操作类、计算类。对每一个关键类，都可以开发其支持类。在演化过程中，支持类是迭代定义的。支持类的数量既是开发软件所需工作量的指标，也是系统开发中潜在的复用数量的指标。

每个关键类的平均支持类数量。通常，关键类在项目的早期就可以确定下来，而支持类的定义则贯穿于项目的始终。对于给定的问题域，如果知道了每个关键类的平均支持类数量，估算（根据类的总数）就将变得极其简单。Lorenz 和 Kidd 指出，在采用 GUI 的应用中，支持类是关键类的 2 ~ 3 倍；在不采用 GUI 的应用中，支持类是关键类的 1 ~ 2 倍。

子系统的数量。子系统是实现某个功能（对系统最终用户可见）的类的集合。一旦确定了子系统，人们就更容易制定出合理的进度计划，并将子系统的工作在项目人员之间进行分配。

为了将上述这些度量有效地应用于面向对象的软件工程环境中，必须将它们随同项目测量（例如，花费的工作量、发现的错误和缺陷、建立的模型或文档资料）一起收集。随着数据库规模的增长（在完成大量项目之后），面向对象测量和项目测量之间的关系将提供有助于项目估算的度量。

32.2.5 面向用例的度量

用例^②被广泛地用于描述客户层或业务领域的需求，这些需求中隐含着软件的特性和功能。与 LOC 或 FP 类似，使用用例作为规范化的测量应该是合理的。用例同 FP 一样，也是在软件过程早期进行定义。在重大的建模活动和构建活动开始之前，就允许使用用例进行估算。用例描述了（至少是间接地）用户可见的功能和特性，这些都是系统的基本需求。用例与程序设计语言无关。另外，用例的数量同应用的规模（LOC）和测试用例的数量成正比，而测试用例是为了充分测试该应用而必须要设计的。

由于可以在不同的抽象级别上创建用例，所以用例的“大小”没有统一标准。由于对用例本身都没有标准的“测量”，因此将用例作为规范化的测量（例如，每个用例花费的工作

建议 多个场景脚本涉及同一个功能或数据对象，这种情况很常见，因此应该慎重使用脚本数量这个度量。有时可以将多个脚本简化为一个类或一组代码。

建议 每个类的规模和复杂性不同，因此可以考虑将类按照规模和复杂性的不同分别统计其数量。

[713]

① 在第 10 章中，关键类被称为分析类。

② 关于用例的介绍已在第 8 章和第 9 章中给出。

量)是不可信的。

研究人员提议将用例点 (UCP) 作为一种估算项目工作量及其他特性的机制。UCP 是用例模型所包含的参与者数量及业务数量的函数,在某些方面与 FP 相似。如果有兴趣进一步了解,可参见 [Coh05]、[Cle06] 或 [Col09]。

32.2.6 WebApp 项目的度量

714

所有 WebApp 项目的目标都是向最终用户交付内容和功能的结合体。很难将那些用于传统软件工程项目的测量和度量直接转化并应用于 WebApp 中。然而,建立一个数据库,随着大量项目的完成,访问该数据库获得内部的生产率和质量测量,这是可能的。在这些测量中,可以收集的内容如下。

静态 Web 页的数量。静态页面复杂性较低,构建这些页面所需的工作量通常少于动态页面。这项测量提供了一个标志着应用整体规模和开发应用所需工作量的指标。

动态 Web 页的数量。动态页面复杂性较高,构建这些页面所需的工作量高于静态页面。这项测量提供了一个标志着应用整体规模和开发应用所需工作量的指标。

内部页面链接的数量。这项测量提供了一个 WebApp 内部结构互连程度的指标。随着页面链接的增加,花费在导航设计和开发上的工作量也会增加。

永久数据对象的数量。随着永久数据对象(如数据库或数据文件)数量的增长,WebApp 的复杂性增大,实现应用所需的工作量也会成比例增加。

通过界面连接的外部系统的数量。随着界面连接需求的增多,系统的复杂性和开发工作量随之增加。

静态内容对象的数量。这些对象的复杂度相对较低,构建这些对象的工作量通常少于动态页面。

动态内容对象的数量。这些对象的复杂度相对较高,构建这些对象的工作量高于静态页面。

可执行的功能的数量。随着可执行功能(例如脚本或小程序)数量的增长,建模和构建的工作量会随之增加。

上述每个测量在初期就可以确定下来。例如,可以定义一个度量,来反映 WebApp 所需的最终用户的定制程度,并使它与项目花费的工作量以及评审和测试中发现的错误关联起来。为此,做以下定义:

$$N_{sp} = \text{静态 Web 页的数量}$$
$$N_{dp} = \text{动态 Web 页的数量}$$

那么,定制指数定义为:

$$C = \frac{N_{dp}}{(N_{dp} + N_{sp})}$$

C 的取值范围是 0 到 1。随着 C 值的增大,WebApp 的定制水平将成为一个重大的技术

715

问题。

类似的 WebApp 度量也可以计算出来,并同项目测量(例如,花费的工作量、发现的错误和缺陷、已建立的模型或文档)关联起来。随着数据库规模的扩大(在完成了很多项目之

后), WebApp 测量与项目测量之间的关系将提供有助于项目估算的指标。

软件工具 项目度量和过程度量

[目标] 辅助进行软件测量和度量的定义、收集、评估和报告。

[机制] 每个工具在应用上有所不同, 不过所有的工具都提供了一定的方法来收集和评估软件度量计算所用的数据。

[代表性工具]^①

- Function Point WORKBENCH。由 Charismatek (www.charismatek.com.au) 开发, 提供了大量的面向功能点的度量。
- DataDrill。由 Distributive Software (www.distributive.com) 开发, 支持自动化的数据收集、分析、图表格式化、报表生成及其他测量任务。

- PSM Insight。由 Practical Software and Systems Measurement (www.psmssc.com) 开发, 帮助创建项目测量数据库及随后进行的分析。
- SLIM tool set。由 QSM (www.qsm.com) 开发, 提供了一整套的度量和估算工具。
- SPR tool set。由 Software Productivity Research (www.spr.com) 开发, 收集了一整套面向功能点的工具。
- TychoMetrics。由 Predicate Logic (www.predicate.com) 开发的一个工具组, 用来管理度量的收集和报告。

32.3 软件质量的度量

系统、应用或产品的质量取决于描述问题的需求、建模解决方案的设计、导出可执行程序的编码以及执行软件来发现错误的测试。可以使用测量来获知需求与设计模型的质量、源代码的质量以及构建软件时所创建的测试用例的质量。为了做到这种实时的评价, 必须应用产品度量 (第 30 章) 来客观而不是主观地评估软件工作产品的质量。

随着项目的进展, 项目经理也必须评估质量。将软件工程师个人收集的私有度量结合起来, 可以提供项目级的结果。虽然可以收集到很多质量的测量数据, 但在项目级上最主要的还是测量错误和缺陷。从这些测量中导出的度量能够提供一个指标, 表明个人及小组在软件质量保证和控制活动上的效力。

度量, 如每功能点的工作产品错误数、评审时每小时发现的错误数、测试时每小时发现的错误数, 使我们能够深入了解度量所隐含的每项活动的功效。有关错误的数数据也能用来计算每个过程框架活动的缺陷排除效率 (Defect Removal Efficiency, DRE)。DRE 将在 32.3.3 节讨论。

32.3.1 测量质量

虽然有很多关于软件质量的测量指标^②, 但正确性、可维护性、完整性和可用性为项目团队提供了有用的指标。Gilb[Gil88] 分别给出了它们的定义和测量。

建议 软件是一个复杂的实体。随着工作产品的开发, 错误也会产生。过程度量就是要改进软件过程, 以便更有效地发现错误。

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

② 关于影响软件质量的因素和可用于评估软件质量的度量已在第 30 章详细讨论。

网络资源 关于软件质量及相关主题(包括度量)的优秀信息源可参见 <http://searchsoftwarequality.techtarget.com/resources>。

正确性。正确性是软件完成所要求的功能的程度。缺陷(正确性缺失)是指在程序发布后经过了全面使用,由程序用户报告的问题。为了进行质量评估,缺陷是按标准时间段来计数的,典型的时间是一年。最常用的关于正确性的测量是每千行代码(KLOC)的缺陷数,这里的缺陷是指已被证实不符合需求的地方。

可维护性。可维护性是指遇到错误时程序能够被修改的容易程度,环境发生变化时程序能够适应的容易程度,以及用户希望变更需求时程序能够被增强的容易程度。还没有直接测量可维护性的方法,只能采用间接测量。有一种简单的面向时间的度量,称为平均变更时间(Mean-Time-To-Change, MTTC)。平均变更时间包括分析变更请求、设计合适的修改方案、实现变更并进行测试以及把该变更发布给全部用户所花费的时间。

完整性。这个属性测量的是一个系统对安全性攻击(包括偶然的和蓄意的)的抵抗能力。为了测量完整性,必须定义另外两个属性:危险性和安全性。危险性是指一个特定类型的攻击在给定的时间内发生的概率(能够估算或根据经验数据导出)。安全性是指一个特定类型的攻击被击退的概率(能够估算出来或根据经验数据得到)。系统的完整性可以定义为:

$$\text{完整性} = \sum (1 - (\text{危险性} \times (1 - \text{安全性})))$$

例如,假设危险性(发生攻击的可能性)是0.25,安全性(击退攻击的可能性)是0.95,则系统的完整性是0.99(很高);另一方面,假设危险性是0.5,击退攻击的可能性仅是0.25,则系统的完整性只有0.63(低得无法接受)。

可用性。可用性力图对“使用的容易程度”进行量化,可以根据第15章中给出的特性来测量。

在被建议作为软件质量测量的众多因素中,上述4个因素仅仅是一个样本。关于这些内容,在第30章中已经给出了更详细的讨论。

32.3.2 缺陷排除效率

缺陷排除效率(Defect Removal Efficiency, DRE)是在项目级和过程级都有意义的质量度量。质量保证及质量控制活动贯穿于所有过程框架活动中,DRE本质上就是对质量保证及质量控制动作中滤除缺陷的能力的测量。

把项目作为一个整体来考虑时,可按如下方式定义DRE:

$$DRE = \frac{E}{E+D}$$

其中, E 是软件交付给最终用户之前发现的错误数, D 是软件交付之后发现的缺陷数。

DRE最理想的值是1,即在软件中没有发现缺陷。实际上, D 的值大于0,但DRE仍可能接近于1。对于一个给定的 D 值,随着 E 的增加,DRE的整体数值越来越接近于1。实际上,随着 E 的增加, D 的最终值会降低(错误在变成缺陷之前已经被滤除了)。如果将DRE作为一个度量,提供关于质量控制及质量保证活动的滤除能力的衡量指标,那么DRE就能促使软件项目团队采用先进的技术,力求在软件交付之前发现尽可能多的错误。

建议 如果从分析阶段进入设计阶段时,DRE的值较低,你就要花些时间去改进正式技术评审的方式了。

在项目内部,也可以使用DRE来评估一个团队在错误传递到下一个框架活动或软件工

程任务之前发现错误的能力。例如，需求分析创建了一个需求模型，而且对该模型进行了评审来发现和改正其中的错误。那些在评审过程中未被发现的错误传递给了设计（在设计中它们可能被发现，也可能没有被发现）。在这种情况下，我们将 DRE 重新定义为：

$$DRE_i = \frac{E_i}{E_i + E_{i+1}}$$

其中， E_i 是在软件工程动作 i 中发现的错误数； E_{i+1} 是指在软件工程动作 $i+1$ 中发现的而在软件工程动作 i 中没有被发现的错误的数量。

软件团队（或软件工程师个人）的质量目标是使 DRE_i 接近于 1，即错误应该在传递到一个活动或动作之前被滤除。

718

SafeHome 建立度量方法

[场景] Doug Miller 的办公室，在首次召开软件度量会议两天之后。

[人物] Doug Miller, SafeHome 软件团队经理；Vinod Raman 和 Jamie Lazar, 产品软件工程团队成员。

[对话]

Doug: 关于过程度量和项目度量，你们都有所了解了吧。

Vinod 和 Jamie 都点头。

Doug: 无论采用何种度量，都要建立目标，这总是正确的。那么，你们的目标是什么？

Vinod: 我们的度量应该关注质量。实际上，我们的总体目标是使得上一个软件工程活动传递给下一个软件工程活动的错误数最少。

Doug: 并且确保随同产品发布的缺陷数尽可能接近于 0。

Vinod (点头): 当然。

Jamie: 我喜欢将 DRE 作为一个度量。我

认为我们可以将 DRE 用于整个项目的度量。同样，当从一个框架活动转到下一个框架活动时，也可以使用它。DRE 促使我们在每一步都去发现错误。

Vinod: 我觉得还要收集我们用在评审上的小时数。

Jamie: 还有我们花费在每个软件工程任务上的总工作量。

Doug: 可以计算出评审与开发的比率，这可能很有趣。

Jamie: 我还想跟踪一些用例方面的数据，如建立一个用例所需的工作量，构建软件来实现一个用例所需的工作量，以及……

Doug (微笑): 我想我们要保持简单。

Vinod: 应该这样，不过你一旦深入到度量中，就可以看到很多有趣的事。

Doug: 我同意，但在我们会跑之前要先走，坚持我们的目标。收集的数据限制在 5 到 6 项，准备去做吧。

32.4 在软件过程中集成度量

大多数软件开发者还没有进行测量，更可悲的是，他们中的大多数人根本没有开始测量的愿望。正如本章前面所提到的，这是文化的问题。试图收集过去从来没有人收集过的测量数据常常会遇到阻力。备受折磨的项目经理会问：“为什么我们要做这些？”超负荷工作的开发者会抱怨“我看不出这样做有什么用。”

在本节中，我们考虑一些有关软件度量的观点，并给出在软件工程组织内部制定度量收

集计划的方法。不过,在开始前,我们先来看看 Grady 和 Caswell[Gra87]所说的充满智慧的话(距今已近 30 年):

这里描述的一些事情听起来似乎相当容易。但实际上,成功地制定全公司范围内的软件度量大纲是很困难的工作。如果我们说,你必须至少等上 3 年才能在组织内形成显著的趋势,你就能对这一工作量的规模有一定了解了。

作者给出的告诫值得很好地借鉴。但是,测量的作用是如此显著,再艰苦的工作也是值得做的。

32.4.1 支持软件度量的论点

测量软件工程过程及其生产出来的产品(软件)为什么这么重要?答案其实很明显。如果不进行测量,就无法确定你是否在改进。如果你没有在改进,就会导致失败。

通过对生产率测量和质量测量提出请求并进行评估,软件团队(及其管理者)能够建立改进软件工程过程的有意义的目标。本书的开头部分提到,对许多公司而言,软件都是一个战略性的商业产物。如果软件开发过程能够得到改进,对最终结果(bottom line)将产生直接的影响。而要建立改进目标,就必须了解当前的软件开发状态。因此,可以使用测量来建立过程的基线,根据此基线来评估改进。

每天繁重的软件项目工作使得人们几乎没有时间进行战略性的思考。软件项目经理更关心现实的问题(当然这也同样重要)。例如,建立有意义的项目估算、开发高质量的系统、按期交付产品等。通过使用测量来建立项目基线,将使这些问题变得更易于管理。我们已经知道,基线是估算的基础。此外,质量度量的收集可以使一个组织“调整”其软件工程过程,以消除那些对软件开发有重大影响的缺陷产生的根源^①。

引述 在生活的很多方面,我们都是通过“数字”来管理事物……这些数字使我们对于事物有了深入的了解,并有助于指导我们的行动。

Michael Mah,
Larry Putnam

32.4.2 建立基线

通过建立度量基线,在过程级、项目级和产品(技术)级上都能获得收益。而要收集的信息并非完全不同,相同的度量可以用于多个方面。度量基线由以往开发的软件项目中收集的数据构成,它可能像图 32-2 所示的表格那样简单,也可能像一个综合数据库——包含了几十个项目的测量数据以及从中导出的度量——那样复杂。

为了有效地协助过程改进和成本及工作量估算,基线数据必须具有下列属性:(1)数据必须相当精确,要避免对过去项目进行“推测”;(2)应该从尽可能多的项目中收集数据;(3)测量数据必须是一致的(例如,在收集数据涉及的所有项目中对代码行的解释必须是一致的);(4)基线数据所属的应用应该与要做估算的工作类似,如果将一个适用于批处理系统工作的基线用于估算实时嵌入式应用软件,那就没有什么意义了。

提问 什么是度量基线?它能为软件工程师提供什么帮助?

32.4.3 度量收集、计算和评估

建立度量基线的过程如图 32-3 所示。理想情况下,建立基线所需的数据已经在项目开

① 这些想法已经规范化成为一种方法,称为统计软件质量保证。

发过程中收集了。但遗憾的是，很少有人会这样做。因此，在收集数据时，需要对以往的项目做历史调查，重建所需的数据。一旦收集好了测量数据（无疑这是最困难的一步），就可以进行度量计算了。依赖于所收集的测量数据的广度，度量可以涵盖众多面向应用的度量（例如，LOC、FP、面向对象、WebApp）以及其他面向质量和面向项目的度量。最后，度量还要在估算、技术工作、项目控制及过程改进中加以评估和使用。度量评估主要是分析结果产生的根本原因，并生成一组指导项目或过程的指标。

建议 如果你正要开始收集度量数据，记住要保持简单性。如果将自己淹没于数据之中，那么，你的度量工作将注定要失败。

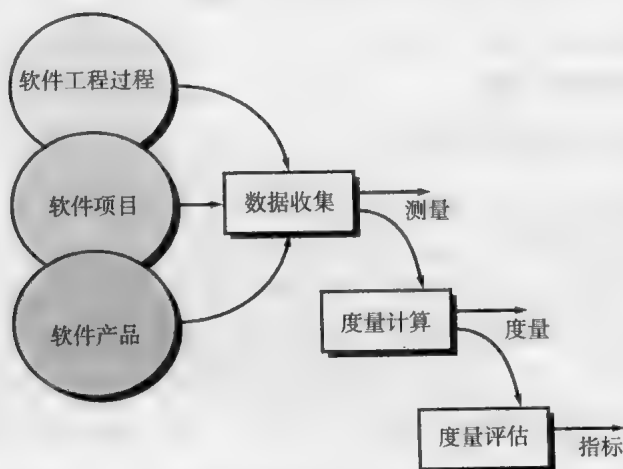


图 32-3 软件度量的收集过程

32.5 小型组织的度量

在绝大多数软件开发组织中，软件人员都不到 20 人。期望这样的组织能制定出全面的软件度量大纲是不合理的，多数情况下也是不现实的。然而，建议各种规模的软件组织^①都进行测量，然后使用从中导出的度量来帮助他们改进其软件过程，提高所开发产品的质量，缩短开发时间，这样的要求是合理的。

小型组织一开始先不要关注测量，而是要从结果入手。软件小组通过表决确定一个需要改进的目标，例如，“减少评估和实现变更请求的时间”。根据这个目标，小型组织可以选择下列易于收集的测量：

- 从提出请求到评估完成所用的时间（小时或天）， t_{queue} 。
- 进行评估所用的工作量（人时）， W_{eval} 。
- 从完成评估到把变更工单派发到员工所用的时间（小时或天）， t_{eval} 。
- 实现变更所需的工作量（人时）， W_{change} 。
- 实现变更所需的时间（小时或天）， t_{change} 。
- 在实现变更过程中发现的错误数， E_{change} 。
- 将变更发布给客户后发现的缺陷数， D_{change} 。

提问 我们应该怎样导出一组“简单的”软件度量？

一旦从大量变更请求中收集到了这些测量数据，就能计算出从变更请求到变更实现所用的总时间，以及初始排队、评估、派发变更和实现变更所占用时间的百分比。类似地，还

① 对于已经采用了敏捷软件开发过程（第 5 章）的团队来说，这项讨论同样具有指导意义。

可以计算出评估和实现变更所需工作量的百分比。这些度量也可以根据质量数据 E_{change} 和 D_{change} 来评估。从这些百分比数据还可以清楚地看出变更请求过程在什么地方延迟了,从而进行过程改进,以减少 t_{queue} 、 W_{eval} 、 t_{eval} 、 W_{change} 和 E_{change} 。此外,缺陷排除效率又可以用以下公式计算:

$$\text{DRE} = \frac{E_{\text{change}}}{E_{\text{change}} + D_{\text{change}}}$$

将 DRE 同变更所用的时间和总工作量进行比较,可以看出质量保证活动对变更所需时间和工作量的影响。

32.6 制定软件度量大纲

美国卡内基·梅隆大学软件工程研究所 (SEI) 已经开发了一套用于制定“目标驱动的”软件度量大纲的综合指导手册 [Par96b]。手册中给出了以下步骤:(1) 明确你的业务目标;(2) 厘清你要了解或学习的内容;(3) 确定你的子目标;(4) 确定与子目标相关的实体和属性;(5) 确定你的测量目标;(6) 识别可量化的问题和相关的指标,你将使用它们帮助你达到测量目标;(7) 明确你要收集的为构成指标所包含的数据元素;(8) 定义将要使用的测量,这些定义要具有可操作性;(9) 清楚实现测量需要做的操作;(10) 准备一份实施测量的计划。关于这些步骤的详细讨论最好参见 SEI 的手册。不过,有必要简要阐述一下关键问题。

因为软件支持业务功能,它分为基于计算机的系统或产品,或者本身就是产品,因此针对业务所定义的目标几乎总是可以向下追溯到软件工程层次上的特定目标。例如,考虑 SafeHome 产品。软件工程师和业务管理者协同工作,制定出一组按优先级排列的业务目标:

1. 提高客户对产品的满意度。
2. 使产品易于使用。
3. 缩短将新产品推向市场的时间。
4. 使产品支持更容易。
5. 提高整体收益率。

软件组织审查每个业务目标并提出问题:“我们管理、执行或支持什么活动?在这些活动中我们要改进什么?”为了回答这些问题,SEI 建议创建一个“实体-问题表”。在这个表中,列出所有在软件过程中受软件组织管理或影响的事物(实体)。实体的例子包括开发资源、工作产品、源代码、测试用例、变更请求、软件工程任务及进度安排。对列出的每个实体,软件人员都要提出一组评估其定量特征(例如,大小、成本、开发时间)的问题。创建实体-问题表引出了这些问题,从而又导出了一组子目标,这些子目标与已创建的实体和已完成的部分软件过程活动紧密相关。

考虑第 4 个目标:“使产品支持更容易。”由这个目标可以引出下列问题 [Par96b]:

- 客户的变更请求中包含及时评估变更并实现变更所需要的信息吗?
- 积压的变更请求有多少?
- 根据客户的需要,我们修正缺陷的响应时间能接受吗?
- 遵循变更控制过程(第 29 章)吗?

关键点 你选择的软件度量应该由你所希望达到的业务和技术目标来驱动。

● 优先级高的变更能及时实现吗？

根据这些问题，软件组织可以导出下面的子目标：提高变更管理过程的效能。然后，确定与该子目标相关的软件过程实体和属性，明确与这些实体和属性相关的测量目标。

723

SEI[Par96b] 对目标驱动测量方法中的步骤 6 ~ 10 给出了详细指导。本质上是将测量目标细化为问题，这些问题进一步细化为实体和属性，然后将这些实体和属性细化为度量。

32.7 小结

测量能使管理者和开发者改进软件过程，辅助进行软件项目的计划、跟踪及控制，评估所生成的产品（软件）的质量。对过程、项目及产品的特定属性的测量可用来计算软件度量。分析这些度量可以获得指导管理及技术行为的指标。

过程度量能使一个组织从战略角度深入了解软件过程的功效。项目度量是战术性的，能使项目管理者实时地改进项目的工作流程及技术方法。

面向规模的度量和面向功能的度量在业界都得到了广泛应用。面向规模的度量以代码行作为其他测量（如人月或缺陷）的规范化因子。功能点则是从信息域测量及对问题复杂度的主观评估中导出的。此外，还能使用面向对象的度量和 WebApp 度量。

软件质量度量（如生产率度量）关注的是过程、项目和产品。一个组织通过建立并分析质量度量基线，能够纠正那些引起软件缺陷的软件过程区域。

测量会带来企业文化的改变。如果开始进行度量，那么数据收集、度量计算和度量分析是必须完成的三个步骤。通常，目标驱动方法有助于一个组织关注自身业务的正确度量。通过建立度量基线——一个包含过程和产品测量的数据库，软件工程师及其管理者能够更好地了解他们所做的工作和开发的产品。

习题与思考题

32.1 用自己的话描述过程度量和项目度量之间的区别。

32.2 为什么有些软件度量是“私有的”？给出 3 个私有度量的例子，并给出 3 个公有度量的例子。

32.3 什么是间接测量？为什么在软件度量工作中经常用到这类测量？

724

32.4 Grady 提出了一组软件度量规则，你能在 32.1.1 节所列的规则中再增加 3 个规则吗？

32.5 产品交付之前，团队 A 在软件工程过程中发现了 342 个错误，团队 B 发现了 184 个错误。对于项目 A 和 B，还需要做什么额外的测量，才能确定哪个团队能够更有效地排除错误？你建议采用什么度量来帮助做出判定？哪些历史数据可能有用？

32.6 给出反对将代码行作为软件生产率度量的论据。当考虑几十个或几百个项目时，你说的情况还成立吗？

32.7 根据下面的信息域特性，计算项目的功能点值：

用户输入数：32

用户输出数：60

用户查询数：24

文件数：8

外部接口数：2

假定所有的复杂度校正值都取“中等”值。使用第 30 章描述的算法。

32.8 利用 32.2.3 节中给出的表格，基于每行代码具有的功能性，提出一个反对使用汇编语言的论据。再参考该表，讨论为什么 C++ 比 C 更好。

- 32.9 用于控制影印机的软件需要 32000 行 C 语言代码和 4200 行 Smalltalk 语言代码。估算该影印机软件的功能点数。
- 32.10 某 Web 工程团队已经开发了一个包含 145 个网页的电子商务 WebApp。在这些页面中, 有 65 个是动态页面, 即根据最终用户的输入而在内部生成的页面。那么, 该应用的定制指数是多少?
- 32.11 一个 WebApp 及其支持环境没有被充分地加强来抵御攻击。Web 工程师估计击退攻击的概率只有 30%。系统不包含机密或有争议的信息, 因此危险性概率只有 25%。那么, 该 WebApp 的完整性是多少?
- 32.12 在一个项目结束时, 确定在建模阶段发现了 30 个错误, 在构建阶段发现了 12 个错误, 这 12 个错误可以追溯到建模阶段没有发现的错误。那么, 这两个阶段的 DRE 是多少?
- 32.13 软件团队将软件增量交付给最终用户。在第一个月的使用中, 用户发现了 8 个缺陷。在交付之前, 软件团队在正式技术评审和所有测试任务中发现了 242 个错误。那么在使用一个月之后, 项目总的缺陷排除效率 (DRE) 是多少?

扩展阅读与信息资源

在过去的 20 年中, 软件过程改进 (SPI) 受到了极大的关注。由于测量和软件度量是成功改进软件过程的关键, 所以在很多 SPI 方面的书籍中也讨论度量。Arban (《Software Metrics and Software Methodology》, Wiley-IEEE Computer Society, 2010) 和 Rico (《ROI of Software Process Improvement》, J. Ross Publishing, 2004) 所著的书深入讨论了 SPI 以及能够帮助组织进行过程改进的度量。Ebert 及其同事 (《Best Practices in Software Measurement》, Springer, 2004) 在 ISO 和 CMMI 标准的范畴内讨论了测量的使用。Kan (《Metrics and Models in Software Quality Engineering》, 2nd ed., Addison-Wesley, 2002) 介绍了相关度量的收集。

Ebert 和 Dumke (《Software Measurement》, Springer, 2007) 提供了将测量和度量应用于 IT 项目时的有用的处理措施。McGarry 及其同事 (《Practical Software Measurement》, Addison-Wesley, 2001) 对评估软件过程提出了深层次的建议。Haug 及其同事已经编辑了一部值得收藏的论文集 (《Software Process Improve: Metrics, Measurement, and Process Modeling》, Springer-Verlag, 2001)。Florac 和 Carlton (《Measuring the Software Process》, Addison-Wesley, 1999) 以及 Fenton 和 Pfleeger (《Software Metrics: A Rigorous and Practical Approach》, Revised, Brooks/Cole Publishers, 1998) 探讨了如何利用软件度量来取得改进软件过程的必要指标。

Wohlin 和他的同事 (《Experimentation in Software Engineering》, Springer, 2012) 讨论了用于分析软件过程的测量的使用方式。Jones (《Applied Software Measurement: Global Analysis of Productivity and Quality》, McGraw-Hill, 2008)、Laird 和 Brennan (《Software Measurement and Estimation》, Wiley-IEEE Computer Society Press, 2006) 以及 Goodman (《Software Metrics: Best Practices for Successful IT Management》, Rothstein Associates, 2004) 讨论了如何将软件度量用于项目管理和估算。Putnam 和 Myers (《Five Core Metrics》, Dorset House, 2003) 利用一个包含 6000 多个软件项目的数据库, 论证了如何使用 5 种核心度量——时间、工作量、规模、可靠性以及过程生产率——来控制软件项目。Maxwell (《Applied Statistics for Software Managers》, Prentice-Hall, 2003) 给出了分析软件项目数据的技术。Munson (《Software Engineering Measurement》, Auerbach, 2003) 讨论了大量的软件工程测量问题。Jones (《Software Assessments, Benchmarks and Best Practices》, Addison-Wesley, 2000) 描述了定量的测量以及定性的测量因素, 帮助组织评估自身的软件过程和实践。

功能点测量已经成为一种广泛应用于软件工程工作很多领域的技术。国际功能点用户组出版了关于使用功能点度量的论文集（《The IFPUC Guide to IT and Software Measurement》，Auerbach, 2012）。Parthasarathy（《Practical Software Estimation：Function Point Methods for Insourced and Outsourced Projects》，Addison-Wesley, 2007）提供了综合性的指南。Garmus 和 Herron（《Function Point Analysis：Measurement Practices for Successful Software Projects》，Addison-Wesley, 2000）讨论了侧重于功能点分析的过程度量。

关于 Web 工程工作的度量，已发表的资料比较少，不过 Clifton（《Advanced Web Metrics with Google Analytics》，3rd ed., Sybex, 2012）、Kaushik（《Web Analytics 2.0：Accountability and Science of Customer Centricity》，Sybex, 2009 和《Web Analytics：An Hour a Day》，Sybex, 2007）、Stern（《Web Metrics：Proven Methods for Measuring Web Site Success》，Wiley, 2002）、Inan 和 Kean（《Measuring the Success of Your Website》，Longman, 2002）以及 Nobles 和 Grady（《Web Site Analysis and Reporting》，Premier Press, 2001）等人的著作从商业和市场角度讨论了 Web 度量。

IEEE 总结了度量领域的最新研究（《Symposium on Software Metrics》，每年出版）。大量的关于过程度量和项目度量的信息源可以在网上获得。最新的参考文献参见 SEPA 网站 www.mhhe.com/pressman 下的“software engineering resources”。

软件项目估算

要点浏览

概念：软件的真实需求已经确定；利益相关者都已到位；软件工程师准备开始工作；项目将要启动。但是如何进行下去呢？软件项目计划包括 5 项主要活动——估算、进度安排、风险分析、质量管理计划和变更管理计划。在本章中，我们只考虑估算——尝试确定构建一个特定的基于软件的系统或产品所需投入的资金、工作量、资源及时间。

人员：软件项目经理——利用从项目利益相关者那里获得的信息以及从以往项目中收集的软件度量数据。

重要性：你会在不知道要花多少钱、要完成多少任务以及完成工作需要多少时间的情况下建造房子吗？当然不会。既然大多数基于计算机的系统和产品的成本大大超过建造一所大房子，那么在开发

软件之前进行估算应该是合理的。

步骤：估算首先要描述产品的范围，然后将问题分解为一组较小的问题，再以历史数据和经验为指南，对每个小问题进行估算。在进行最终的估算之前，要考虑问题的复杂度和风险。

工作产品：生成一个简单的表，描述要完成的任务和要实现的功能，以及完成每一项所需的成本、工作量和时间。

质量保证措施：这很困难。因为一直要等到项目完成时，你才能真正知道。不过，如果你有经验并遵循系统化的方法，使用可靠的历史数据进行估算，利用至少两种不同的方法创建估算数据点，制定现实的进度表并随着项目的进展不断进行调整，那么你就可以确信你已经为项目做了最好的估算。

软件项目管理从一组统称为项目计划的活动开始。在项目启动之前，软件团队应该估算将要要做的工作、所需的资源，从开始到完成所需要的时间。这些活动一旦完成，软件团队就应该制定项目进度计划。在项目进度计划中，要定义软件工程任务及里程碑，指定每一项任务的负责人，详细说明对项目进展有较大影响的任务间的相互依赖关系。

在一本关于“软件项目生存”的优秀指南中，Steve McConnell [McC98] 讲述了人们对项目计划的实际看法：

很多技术工作者宁愿从事技术工作，也不愿花费时间制定计划。很多技术管理者没有受过充分的技术管理方面的培训，对他们的计划是否能够改善项目成果缺乏信心。既然这两部分人都不想制定计划，因此就经常不制定计划。

关键概念

估算

敏捷开发

分解技术

经验模型

基于功能点

自制 - 外购

面向对象的项
目

外包

基于问题

基于过程

但是，没有很好地制定计划是一个项目犯的最严重的错误之一……有效的计划是必需的，可以在上游（项目早期）以较低的成本解决问题，而不是在下游（项目后期）以较高的成本解决问题。一般的项目要将 80% 的时间花费在返工上——改正项目前期所犯的错误。

McConnell 指出，每个项目都能找出制定计划的时间（并使计划适应于整个项目），只要从因为没制定计划而出现的返工时间中抽出一小部分时间即可。

33.1 对估算的观察

制定计划需要你做一个初始约定，即使这个“约定”很可能被证明是错误的。无论在什么时候进行估算，都是在预测未来，自然要接受一定程度的不确定性。下面引用 Frederick Brooks[Bro95] 的话：

……我们的估算技术发展缓慢。更为严重的是，它们隐含了一个很不正确的假设，即“一切都会好的”……因为对自己的估算没有把握，软件管理者常常缺乏做出好产品的信心。

估算是一门艺术，更是一门科学，这项重要的活动不能以随意的方式进行。现在已经有了估算时间和工作量的实用技术。过程度量和项目度量定量估算提供了历史依据和有效输入。当建立估算和评审估算时，以往的经验（包括所有参与人员的经验）具有不可估量的辅助作用。由于估算是所有项目计划活动的基础，而项目计划提供了通往成功的软件工程的路线图。因此，没有估算就着手开发将会使我们陷入盲目。

对软件工程工作的资源、成本及进度进行估算时，需要经验，需要了解有用的历史信息（度量）。当只存在定性的信息时，还要有进行定量预言的勇气。估算具有与生俱来的风险[⊖]，正是这种风险导致了不确定性。

项目的复杂性对计划固有的不确定性有很大影响。但是，复杂性是一个相对量，受人员在以往工作中对它的熟悉程度的影响。一个高级的电子商务应用对于首次承担开发工作的程序员来说可能极其复杂。但是，当 Web 工程团队第十次开发电子商务 WebApp 时，会认为这样的工作很普通。现在已经提出了很多软件复杂性的定量测量方法 [Zus97]。这些测量方法都应用于设计层或代码层，因此在软件策划（先于设计和代码的存在）期间难以使用。但是，其他更主观的复杂性评估方法（例如，第 30 章中描述的功能点复杂度校正系数）可以在早期的策划过程中建立。

项目规模是另一个能影响估算精确度和功效的重要因素。随着规模的扩大，软件各个元素之间的相互依赖迅速上升[⊖]。问题分解作为一种重要的估算方法，会由于问题元素的细化仍然难以完成而变得更加困难。用 Murphy 法则来解释：“凡事只要有可能出错，那就一定会出错。”这意味着如果有更多事情可能失败，则这些事情一定失败。

关键概念

估算
调和
用例
WebApp
项目计划
资源
软件方程
软件范围
软件规模估算
用例点 (UCP)

引述 良好的估算

方法和可靠的历史数据提供了最好的希望：现实将战胜不可能的要求。

Caper Jones

关键点 项目复杂性、项目规模以及结构的不确定程度都影响着估算的可靠性。

728

⊖ 系统的风险分析技术将在第 35 章讨论。

⊖ 当问题的需求改变时，由于“范围的蔓延”使得问题的规模常常会扩大。而项目规模的扩大对项目的成本和进度有几何级数的影响 (Michael Mah, personal communication)。

结构的不确定程度也影响着估算的风险。这里，结构是指需求已经被固化的程度、功能被划分的容易程度以及必须处理的信息的层次特性。

历史信息的有效性对估算的风险有很大影响。通过回顾过去，你能仿效做过的工作，并改进出现问题的地方。如果能取得以往项目的全面软件度量（第32章），估算会有更大的保证，合理安排进度以避免重走过去的弯路，总体风险就会降低。

估算的风险取决于资源、成本及进度的定量估算中存在的 uncertainty。如果对项目范围不够了解，或者项目需求经常改变，不确定性和估算风险就会非常高。作为计划人员，你和客户都应该认识到经常改变软件需求意味着成本和进度上的不稳定性。

不过，你不应该被估算所困扰。现代软件工程方法（例如，演化过程模型）采用迭代开发方法。在这类方法中，当客户改变需求时，应该能够（尽管并不总是易于接受）重新审查估算（在了解更多信息后），并进行修正。

引述 应该满足于事物本性所能容许的精确度，当只可能近似于准确时，不要去寻求绝对的准确。

Aristotle

33.2 项目计划过程

软件项目计划的目标是提供一个能使管理人员对资源、成本及进度做出合理估算的框架。此外，估算应该尝试定义“最好的情况”和“最坏的情况”，使项目的结果能够限制在一定范围内。项目计划是在计划任务中创建的，尽管它具有与生俱来的不确定性，软件团队还是要根据它来着手开发。因此，随着项目的进展，必须不断地对计划进行调整和更新。在下面几节中，将讨论与软件项目计划有关的每一项活动。

建议 你了解的越多，就估算得越好。因此，随着项目的进展要对估算进行更新。

729

任务集 项目计划任务集

1. 规定项目范围
2. 确定可行性
3. 分析风险（第35章）
4. 确定需要的资源
 - a. 确定需要的人力资源
 - b. 确定可复用的软件资源
 - c. 识别环境资源
5. 估算成本和工作量
 - a. 分解问题
 - b. 使用规模、功能点、过程任务或用例等方法进行两种以上的估算。
 - c. 调和不同的估算
6. 制定项目进度计划（第34章）
 - a. 建立一组有意义的任务集合
 - b. 定义任务网络
 - c. 使用进度计划工具制定时间表
 - d. 定义进度跟踪机制

33.3 软件范围和可行性

软件范围描述了将要交付给最终用户的功能和特性、输入和输出数据、作为使用软件的结果呈现给用户的“内容”，还界定了系统的性能、约束条件、接口和可靠性。定义范围可以使用两种技术：

1. 在与所有利益相关者交流之后，写出软件范围的叙述性描述。
2. 由最终用户开发的一组用例。[⊖]

在开始估算之前，首先要对范围陈述（或用例）中描述的功能进行评估，在某些情况下，还要进行细化，以提供更多的细节。由于成本和进度的估算都是面向功能的，因此一定程度的功能分解常常是有益的。性能方面的考虑包括处理时间和响应时间的需求。约束条件表示外部硬件、可用存储或其他现有系统对软件的限制。

一旦确定了软件范围（并征得用户的同意），人们自然会问：我们能够开发出满足范围要求的软件吗？这个项目可行吗？软件工程师常常匆忙越过这些问题（或是被不耐烦的管理者或其他利益相关者催促着越过这些问题），不料竟会一开始就注定要陷入这个项目的泥潭中。

Putnam 和 Myers 建议，只确定范围还不够。一旦理解了范围，就必须进一步确定在可用的技术、资金、时间和资源的框架下，所标识的范围是否能够完成。这是估算过程至关重要的部分，但常常被忽略。

建议 项目可行性很重要，但是，考虑商业需要更重要。构建一个没人想要的高科技系统或产品根本没有意义。

730

33.4 资源

项目计划的第二个任务是对完成软件开发工作所需的资源进行估算。图 33-1 描述了三类主要的软件工程资源——人员、可复用的软件构件及开发环境（硬件和软件工具）。对每类资源，都要说明以下四个特征：资源描述、可用性说明、何时需要资源以及使用资源的持续时间。最后两个特性可以看成是时间窗口。对于一个特定的时间窗口，必须在最早的使用时间建立资源的可用性。

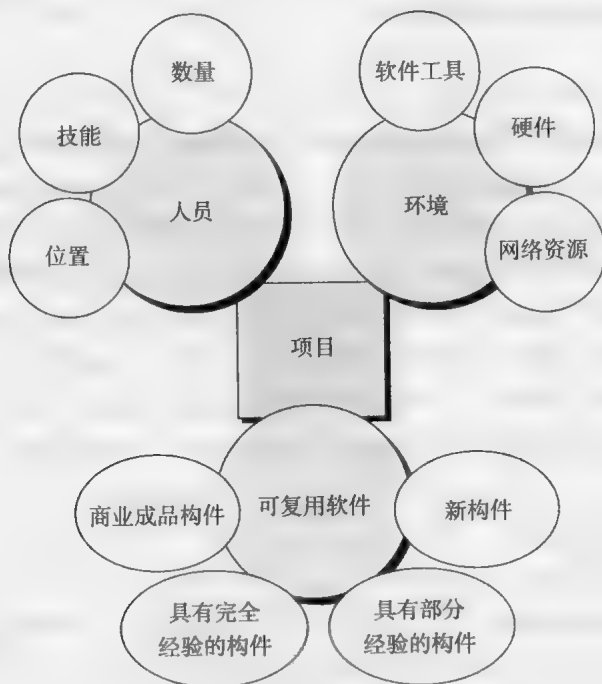


图 33-1 项目资源

⊖ 在本书的第二部分中，已经对用例进行了详细的讨论。用例从用户的观点出发来描述用户与软件的交互场景。

33.4.1 人力资源

计划人员首先评估软件范围，选择完成开发所需的技能，还要指定组织中的职位（例如，管理人员、高级软件工程师）和专业（例如，电信、数据库、客户/服务器系统）。对于一些比较小的项目（几个人月），只要向专家做些咨询，或许一个人就可以完成所有的软件工程任务。而对于一些较大的项目，软件团队的成员可能分散在多个不同的地方，因此，要详细说明每个人所处的位置。

只有在估算出开发工作量（如多少人月）后，才能确定软件项目需要的人员数量。估算工作量的技术将在本章后面讨论。

33.4.2 可复用软件资源

基于构件的软件工程（CBSE）^①强调可复用性，即创建并复用软件构件块，这种构件块通常称为构件。为了容易引用，必须对这些构件进行分类；为了容易应用，必须使这些构件标准化；为了容易集成，必须对这些构件进行确认。Bennatan[Ben00] 建议在制定计划时应该考虑四种软件资源：成品构件（能够从第三方或者从以往项目中获得的现成软件），具有完全经验的构件（为以前项目开发的，具有与当前项目要构建的软件类似的规格说明、设计、代码或测试数据）、具有部分经验的构件（为以前项目开发的，具有与当前项目要构建的软件相关的规格说明、设计、代码或测试数据，但是需要做很多修改），以及新构件（软件团队为了满足当前项目的特定要求，专门开发的软件构件）。

具有讽刺意味的是，在计划阶段，人们往往忽视可复用软件构件，直到软件过程的开发阶段，它才变成最重要的关注对象。最好尽早确定软件资源的需求，这样才能对各种候选方案进行技术评估，及时获取所需的构件。

33.4.3 环境资源

支持软件项目的环境通常称为软件工程环境（Software Engineering Environment, SEE），它集成了硬件和软件。硬件提供支持（软件）工具的平台，而这些（软件）工具是采用良好的软件工程实践来获得工作产品所必需的^②。由于在大多数软件组织中，很多人都需要使用SEE，因此必须详细规定需要硬件和软件的时间窗口，并且验证这些资源是可用的。

当软件团队构建基于计算机的系统（集成特定的硬件和软件）时，可能需要使用其他工程团队开发的硬件元素。例如，在为制造单元中使用的机器人装置开发软件时，可能需要特定的机器人（例如，机器人焊接工）作为确认测试步骤的一部分；在开发高级排版软件项目的过程中，在某个阶段可能需要一套高速数字印刷系统。作为计划的一部分，必须指定每一个硬件元素。

33.5 软件项目估算

软件的成本及工作量估算从来都没有成为一门精确的科学。因为变化的因素太多——人员、技术、环境和行政，都会影响软件的最终成本和开发所用的工作量。不过，软件项目估算还是能够从一种“神秘的”技巧变成一系列系统化的步骤，在可接受的风险范围内提供估

① 第14章讨论了CBSE。

② 其他硬件（目标环境）是指在软件交付给最终用户之后，将要运行该软件的计算机。

[731]

[732]

算结果。为得到可靠的成本和工作量估算，我们有很多选择：

1. 把估算推迟到项目的后期进行（显然，在项目完成之后就能得到 100% 精确的估算）。
2. 根据已经完成的类似项目进行估算。
3. 使用比较简单的分解技术，生成项目的成本和工作量估算。
4. 使用一个或多个经验模型来进行软件成本和工作量的估算。

引述 在外包和竞争愈加激烈的时代，更准确地进行估算的能力……已经成为很多 IT 组织成功的关键因素。

Rob Thomsett

引述 不采用定量的方法、几乎没有数据支持、主要由管理人员的直觉来保证，这样就很难制定有效、可靠、防御工作风险的估算。

Fred Brooks

遗憾的是，不论多吸引人，第一种选择都是不现实的。成本估算必须“预先”给出。不过，应该认识到，你等待的时间越久，了解的就越多。而了解的越多，在估算中出现严重错误的可能性就越小。

如果当前项目与以前的工作非常相似，并且项目的其他影响因素（例如，客户、商业条件、软件工程环境、交付期限）也大致相同，第二种选择就能很好地发挥作用。遗憾的是，过去的经验并不总是能够指明未来的结果。

余下的两种选择对于软件项目估算也是可行的方法。理想的情况是，同时使用这两种选项所提到的技术，相互进行交叉检查。分解技术采用“分而治之”的方法进行软件项目估算，把项目分解成若干主要功能和相关的软件工程活动，以逐步求精的方式对成本和工作量进行估算。经验估算模型可以作为分解技术的补充，在它适用的范围内常常是一种有潜在价值的估算方法。一个基于经验（历史数据）的模型形式如下：

$$d = f(v_i)$$

其中， d 是很多估算值（例如，工作量、成本、项目持续时间）中的一种， v_i 是所选的独立参数（例如，被估算的 LOC 或 FP）。 733

自动的估算工具实现了一种或多种分解技术或经验模型，提供了有吸引力的估算选择。使用这样的系统进行估算时，要描述开发组织的特性（如经验、环境）和待开发的软件，再由这些数据导出成本和工作量估算。

对于每种可行的软件成本估算方法，其效果的好坏取决于估算所使用的历史数据。如果没有历史数据，成本估算就建立在了不稳定的基础上。在第 32 章中，我们已经考察了一些软件度量的特性，这些度量提供了历史估算数据的基础。

33.6 分解技术

软件项目估算是解决问题的一种方式，在多数情况下，要解决的问题（对于软件项目来说，就是成本和工作量估算）非常复杂，不能作为一个整体考虑。因此，要对问题进行分解，把它分解成一组较小的（同时有望更容易管理的）问题，再定义它们的特性。

在第 31 章中，我们从两个不同的角度讨论了分解方法：问题分解和过程分解。估算时可以使用其中一种或两种分解形式。但在进行估算之前，必须理解待开发软件的范围，并估计其“规模”。

33.6.1 软件规模估算

软件项目估算的准确性取决于许多因素：（1）估算待开发产品的规模的正确程度；（2）把规模估算转换成人员工作量、时间及成本的能力（受可靠软件度量的可用性的影响，

这些度量数据来自以往的项目); (3) 项目计划反映软件团队能力的程度; (4) 产品需求的稳定性和支持软件工程工作的环境。

由于项目估算的准确程度取决于待完成工作的规模估算, 因此规模估算是计划人员面临的第一个主要挑战。在项目计划中, 规模是指软件项目的可量化结果。如果采用直接的方法, 规模可以用代码行 (LOC) 来测量。如果选择间接的方法, 规模可以用功能点 (FP) 来表示。规模的估算要考虑项目类型以及应用领域类型、要交付的功能 (如功能点数量)、要交付的构件数量、对现有构件做适用于新系统的修改的程度。

关键点 待开发软件的规模可以使用直接测量 LOC 或间接测量 FP 来估算。

Putnam 和 Myers 建议, 可以将上述每种规模估算方法所产生的结果在统计意义上结合起来, 产生一个三点估算或期望值估算结果。实现方法是: 先确定规模的乐观值 (低)、可能值和悲观值 (高), 然后使用 33.6.2 节的式 (33.1) 将它们结合起来。

33.6.2 基于问题的估算

在第 32 章中, 已经描述了代码行和功能点测量, 从中可以计算出生产率度量。在软件项目估算中, LOC 和 FP 数据用于两个方面: (1) 作为估算变量, 度量软件中每个元素的规模; (2) 作为基线度量, 这些度量数据是从以前的项目中收集起来的, 将它们与估算变量结合使用, 进行成本和工作量的估算。

LOC 估算和 FP 估算是两种不同的估算技术, 但两者有很多相同特性。首先从界定的软件范围陈述入手, 尝试将范围陈述分解成一些可分别独立进行估算的功能问题。然后, 估算每个功能的 LOC 或 FP (即估算变量)。当然, 也可以选择其他元素进行规模估算, 例如, 类或对象、变更、受影响的业务过程。

提问 基于 LOC 和基于 FP 的估算有什么共同点?

然后, 将基线生产率度量 (例如, LOC/pm 或 FP/pm[⊖]) 应用于适当的估算变量, 导出每个功能的成本或工作量。将所有功能的估算合并起来, 即可得到整个项目的总体估算。

然而, 应该注意到, 对于一个组织而言, 其生产率度量常常是变化的, 使用单一基线的生产率度量并不可信。一般情况下, 平均 LOC/pm 或 FP/pm 应该根据项目领域来计算。即应该根据项目的团队规模、应用领域、复杂性以及其他相关参数对项目进行分类, 然后计算出本领域的生产率平均值。估算一个新项目时, 首先应将项目对应到某个领域中, 然后再使用适当的领域生产率平均值对其进行估算。

建议 在收集项目的生产率度量时, 一定要划分项目类型。这样才能计算出特定领域的平均值, 从而使估算更精确。

在分解应用问题时, LOC 估算技术和 FP 估算技术所要求的详细程度及划分目标有所不同。当 LOC 用作估算变量时, 分解是绝对必要的, 而且常常要达到非常详细的程度。分解的程度越高, 就越有可能得到非常精确的 LOC 估算。

对于 FP 估算, 分解则是不同的。它关注的不是功能, 而是 5 个信息域特性——输入、输出、数据文件、查询和外部接口, 以及第 30 章讨论过的 14 种复杂度校正值。然后, 利用这些估算结果导出 FP 值, 该值可与过去的数据结合起来产生估算。

不管使用哪一种估算变量, 你都应该首先为每个功能或每个信息域值确定一个估算值的

⊖ 缩写 pm 代表工作量的单位——人月 (person-month)。

范围。利用历史数据或凭直觉（当其他方法都失效时），为每个功能或每个信息域的计数值都分别估算出一个乐观的、可能的和悲观的规模值。在确定了值的范围后，就得到了一个不确定程度的隐含指标。

接着，计算三点（估算值）或期望值。可以通过乐观值（ S_{opt} ）、可能值（ S_m ）和悲观值（ S_{pess} ）估算的加权平均值来计算估算变量（规模） S 的期望值，例如：

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6} \quad (33.1)$$

其中，“可能”估算值的权重最大，并遵循 β 概率分布。我们假定实际的规模结果落在乐观值与悲观值范围之外的概率很小。

一旦确定了估算变量的期望值，就可以应用历史的 LOC 或 FP 生产率数据。这个估算正确吗？对于这个问题唯一合理的答案就是：“我们不能保证”。对于任何估算技术，不管它有多先进，都必须与其他方法进行交叉检查。即便如此，常识和经验还是会占优势。

33.6.3 基于 LOC 估算的实例

作为 LOC 和 FP 基于问题估算技术的实例，我们考虑为机械零件计算机辅助设计（CAD）应用开发的软件包。该软件将在桌面工作stations上运行，且必须与各种计算机外部绘图设备有接口，包括鼠标、数字化仪、高分辨率彩色显示器和激光打印机。可以给出初步的软件范围陈述：

机械 CAD 软件接受工程师输入的二维或三维几何数据。工程师通过用户界面与 CAD 系统进行交互并控制它，该用户界面应表现出良好的人机界面设计特征。所有的几何数据和其他支持信息都保存在一个 CAD 数据库中。要开发一些设计分析模块，以产生所需的输出，这些输出要显示在各种不同的设备上。软件必须能够控制外部设备（包括鼠标、扫描仪、激光打印机和绘图机），并能与外部设备进行交互。

上述关于范围的陈述是初步的——它没有规定边界。必须对每个句子进行补充说明，以提供具体的细节及定量的边界。例如，在开始估算之前，计划人员必须要确定“良好的人机界面设计特征”是什么含义，或“CAD 数据库”的规模和复杂度是怎样的。

假定我们为了进行估算已经做了进一步的细化，确定了该软件包应具有的主要软件功能，如图 33-2 中所示。遵照 LOC 的分解技术，得到如图 33-2 所示的估算表，表中给出了每个功能的 LOC 估算范围。例如，三维几何分析功能的 LOC 估算范围是：乐观值 4600，可能值 6900，悲观值 8600。应用式（33.1），得到三维几何分析功能的期望值是 6800LOC。通过类似的方法也可以得到其他估算。对 LOC 估算这一列求和，就得到了该 CAD 系统的 LOC 估算值是 33200。

回顾历史数据可以看出，这类系统的组织平均生产率是 620LOC/pm。如果一个劳动力的价格是每月 8000 美元，则每行代码的成本约为 13 美元。根据 LOC 估算及历史生产率数据，该项目总成本的估算值是 431000 美元，工作量的估算值是 54 人月^①。

提问 我们如何计算软件规模的“期望值”？

建议 很多现代应用或者驻留在网络上，或者是客户机/服务器体系结构的一部分。因此，要确信你的估算包含了开发“基础设施软件”所需的工作量。

736

建议 不要屈服于诱惑而使用某一结果作为你的项目估算结果。应该再使用其他方法导出另一个结果来。

① 估算单位是千美元和人月。如果给定了估算的精确度要求，则更高的精确度是不必要的，也是不现实的。

功能	LOC 估算
用户接口及控制设备 (UICF)	2300
二维几何分析 (2DGA)	5300
三维几何分析 (3DGA)	6800
数据库管理 (DBM)	3350
计算机图形显示设备 (CGDF)	4950
外部设备控制功能 (PCF)	2100
设计分析模块 (DAM)	8400
总代码行估算	33200

图 33-2 LOC 方法的估算表

SafeHome 估算

[场景] 项目计划开始时, Doug Miller 的办公室。

[人物] Doug Miller, SafeHome 软件工程技术经理; Vinod Raman、Jamie Lazar 及其他产品软件工程团队成员。

[对话]

Doug: 我们需要对这个项目进行工作量估算, 然后还要为第一个增量制定微观进度计划, 为其余的增量制定宏观进度计划。

Vinod (点头): 好, 但是我们还没有定义任何增量。

Doug: 是的, 但这正是我们需要估算的原因。

Jamie (皱着眉): 你想知道这将花费我们多长时间吗?

Doug: 这正是我需要的。首先, 我们要对 SafeHome 软件进行高层次上的功能分解, 接着, 我们必须估算每个功能对应的代码行数, 然后……

Jamie: 等一下! 我们应该怎样来做?

Vinod: 在过去的项目中, 我已经做过了。你从用例入手, 确定实现每个用例所需要的功能, 然后估计每项功能的 LOC 数。最好的方法是让每个人独立去做, 然后比较结果。

Doug: 或者你可以对整个项目进行功能分解。

Jamie: 但那将花费很长时间, 而我们必须马上开始。

Vinod: 不, 事实上这可以在几个小时内完成, 就今天早上。

Doug: 我同意, 我们不能期望有很高的精确性, 只是了解一下 SafeHome 软件的大致规模。

Jamie: 我认为我们应该只估算工作量, 仅此而已。

Doug: 这我们也要做。然后用这两种估算进行交叉检查。

Vinod: 我们现在就去做吧……

33.6.4 基于 FP 估算的实例

基于 FP 估算时, 问题分解关注的不是软件功能, 而是信息域的值。分别对 CAD 软件的输入、输出、查询、文件和外部接口进行估算, 参看图 33-3 给出的表。然后使用第 30 章讨论的技术来计算 FP 的值。为了进行估算, 假定复杂度加权因子都取平均值。图 33-3 给出了估算的结果。

信息域值	乐观值	可能值	悲观值	估算值	加权因子	FP 值
外部输入数	20	24	30	24	4	97
外部输出数	12	15	22	16	5	78
外部查询数	16	22	28	22	5	88
内部逻辑文件数	4	4	5	4	10	42
外部接口文件数	2	2	3	2	7	15
总计						320

图 33-3 估算信息域的值

估算出每一个复杂度加权因子，根据第 30 章所描述的方法计算出复杂度校正因子的值：

因子	值	因子	值
备份和恢复	4	信息域值复杂度	5
数据通信	2	内部处理复杂度	5
分布式处理	0	设计可复用的代码	4
关键性能	4	设计中的转换与安装	3
现有的操作环境	3	多次安装	5
联机数据输入	4	易于变更的应用设计	5
多屏幕输入切换	5	复杂度校正因子	1.17
主文件联机更新	3		

738

最后，得出 FP 的估算值：

$$FP_{estimated} = 总计 \times (0.65 + 0.01 \times \sum F_i) = 375$$

这类系统的组织平均生产率是 6.5FP/pm。如果一个劳动力价格是每月 8000 美元，则每个 FP 的成本约为 1230 美元。根据 FP 估算和历史生产率数据，项目总成本的估算值是 461000 美元，工作量的估算值是 58 人月。

33.6.5 基于过程的估算

最通用的项目估算技术是根据将要采用的过程进行估算，即将过程分解为一组较小的活动、动作和任务，并估算完成每一项所需的工作量。

同基于问题的估算技术一样，基于过程的估算首先从项目范围中抽取 出软件功能。接着给出为实现每个功能所必须执行的一系列框架活动。这 些功能及其相关的框架活动[⊖]可以用表格形式给出，类似于图 33-4 所示。

一旦将问题功能与过程活动结合起来，就可以针对每个软件功能，估 算出完成各个软件过程活动所需的工作量（如人月），这些数据构成了图 33-4 中表格的中心部分。然后，将平均劳动力价格（即成本 / 单位工作量） 应用于每个软件过程活动的估算工作量，就可以估算出成本。

建议 如果时间 允许，可以使用 更细的粒度来指 定图 33-4 中所 示的任务，例如， 将分析分解为若 干主要任务，并 分别估算每一项 任务。

⊖ 为该项目选择的框架活动与第 3 章中讨论的一般性活动有所不同。这些框架活动是客户沟通（CC）、策划、 风险分析、工程和构建 / 发布。

活动→	客户沟通	策划	风险分析	工程		构建发布		客户评估	合计
任务→				分析	设计	编码	测试		
功能↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
合计	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% 工作量	1%	1%	1%	8%	45%	10%	36%		

图 33-4 基于过程的估算表

739 如果基于过程的估算是 不依赖 LOC 或 FP 估算而实现的，那么就 已经有了两种或三种成本与工作量估算，可以 进行比较和调和。如果 两组估算非常一致，则 有理由相信估算是可 靠的。相反，如果这些 分解技术得到的结果 不一致，则必须做进 一步的调查和分析。

33.6.6 基于过程估算的实例

为了说明基于过程估算的使用方法，我们再次考虑 33.6.3 节介绍的 CAD 软件。系统配置和所有软件功能都保持不变，并已在项目范围中说明。

参看图 33-4 中所示的基于过程的估算表，表中对 CAD 软件的每个功能（为了简化做了省略）都给出了其各个软件工程活动的工作量估算（人月）。其中，工程和构建发布活动又被细分为主要的软件工程任务。对客户沟通、策划和风险分析活动，还给出了总工作量的估算，这些数值都列在表格底部的“合计”行中。水平合计和垂直合计为估算分析、设计、编码及测试所需的工作量提供了指标。应该注意到，“前期”的工程任务（需求分析和设计）花费了全部工作量的 53%，说明这些工作相对更重要。

如果平均一个劳动力的价格是每月 8000 美元，则项目总成本的估算值是 368000 美元，工作量的估算值是 46 人月。如果需要的话，每个框架活动或软件工程任务都可以采用不同的劳动力价格分别进行计算。

33.6.7 基于用例的估算

正如本书第二部分中提到的，用例能使软件团队深入地了解软件的范围和需求。一旦开发出用例，就能够将其用于估算软件项目的计划“规模”。但是，建立基于用例的估算方法还面临着挑战 [Smi99]。用例的描述

引述 在使用一种估算之前，最好先去了解这种估算的背景。

Barry Boehm,
Richard Fairley

提问 为什么开发基于用例的估算技术非常困难？

可以采用多种格式和风格,用例表现的是软件的外部视图(用户视图),因此可以在多个不同的抽象级别上建立用例。用例没有标识出它所描述的功能和特性的复杂性,用例不能描述涉及很多功能和特性的复杂行为(如交互)。

尽管有这些限制,但使用类似于功能点计算(第 30 章)的方式来计算用例点(Use Case Point, UCP)还是可能的。

Cohn(Coh05)指出,用例点的计算必须考虑以下特性:

- 系统中用例的数量和复杂性。
- 系统参与者的数量和复杂性。
- 没有写成用例的各种非功能性需求(如可移植性、性能、可维护性)。
- 项目的开发环境(如编程语言、软件团队的积极性)。

首先,评估每个用例,确定其相对复杂性。简单的用例预示着简单的用户界面、单一的数据库、3 个以下的事务以及可用 5 个以下的类实现。一般性的用例预示着比较复杂的用户界面、2 或 3 个数据库,以及 4 到 7 个事务含有 5 到 10 个类。最后,复杂的用例预示着复杂的用户界面,有多个数据库,使用 8 个以上的事务以及 11 个以上的类。采用这些标准评估每一个用例,将每种类型的用例数量分别乘以一个加权系数 5、10 或 15。将加权后的用例数量求和得到总体的未调整用例权重(Unadjusted Use Case Weight, UUCW) [Nun11]。

接着,评估每个参与者。简单的参与者是一个通过 API 进行通信的自动机(另一个系统、一台机器或设备)。一般性的参与者是一个通过协议或数据存储来通信的自动机。复杂的参与者是人,通过图形用户界面或其他人机接口进行通信。使用这些标准评估每个参与者,将每种类型的参与者的数量分别乘以一个加权系数 1、2 或 3,将加权后的参与者数量求和得到总体的未调整的参与者权重(Unadjusted Actor Weight, UAW)。

考虑技术复杂性因子(TCF)和环境复杂性因子(ECF),对这些未调整值进行修改。有 13 个因子用于估算最终的 TCF,8 个因子用于最终的 ECF 的计算[Coh05]。一旦确定了这些值,则以下面的方式来计算最终的用户点值:

$$UCP = (UUCW + UAW) \times TCF \times ECF \quad (33.2) \quad [741]$$

33.6.8 基于用例点估算的实例

在 33.6.3 节介绍的 CAD 软件包括 3 个子系统组:用户界面子系统(包括 UICF)、工程子系统组(包括 2DGA、3DGA 和 DAM 子系统)及基础设施子系统组(包括 CGDF 子系统和 PCF 子系统)。用户界面子系统由 16 个复杂用例描述。工程子系统组由 14 个一般用例和 8 个简单用例描述。基础设施子系统由 10 个简单用例描述。因此,

$$UUCW = (16 \text{ 用例} \times 15) + ((14 \text{ 用例} \times 10) + (8 \text{ 用例} \times 5)) + (10 \text{ 用例} \times 5) = 470$$

对用例进行分析可发现有 8 个简单的参与者,12 个一般的参与者,4 个复杂的参与者。因此,

$$UAW = (8 \text{ 参与者} \times 1) + (12 \text{ 参与者} \times 2) + (4 \text{ 参与者} \times 3) = 44$$

对技术和环境进行评估后,确定 TCF 和 ECF 的值:

$$TCF = 1.04 \quad ECF = 0.96$$

利用式(33.2)可得:

$$UCP = (470 + 44) \times 1.04 \times 0.96 = 513$$

以过去的项目数据为依据,开发小组每个 UCP 生产 85 LOC。这样,CAD 项目的总体

规模估计为 43600LOC。对投入的工作量或者项目工期也可以做类似的计算。

以 620LOC/pm 作为这类系统的平均生产率,一个劳动力价格是每月 8000 美元,则每行代码的成本约为 13 美元。根据用例估算和历史生产率数据,项目总成本的估算值是 552000 美元,工作量的估算值大约是 70 人月。

33.6.9 调和不同的估算方法

在前面章节中讨论的估算技术导出了多种估算方法,必须对这些估算方法进行调和,以得到对工作量、项目工期或成本的一致估算。对 CAD 软件(33.6.3 节)总工作量的估算,最低值是 46 人月(由基于过程的估算方法得出),最高值是 68 人月(由用例估算方法得出)。平均估算值(使用全部 4 种方法)是 56 人月。与平均估算值相比,最低估算值的偏差约为 18%,最高估算值的偏差约为 21%。

如果估算方法所得结果的一致性很差,怎么办呢?对这个问题的回答是,需要对估算所使用的信息进行重新评估。如果不同估算之间的差别很大,一般能够追溯到以下两个原因之一:(1)计划人员没有充分理解或是误解了项目范围;(2)在基于问题的估算技术中所使用的生产率数据不适合本应用,它们已经过时了(因为这些数据已不能正确反映软件工程组织的情况),或者是误用了。应该确定产生差别的原因,再来调和估算结果。

引述 复杂的方法并不一定会产生更精确的估算,尤其是当开发者在估算时加进自己的直觉时。

Philip Johnson
et al.

信息栏 软件项目的自动估算技术

自动估算工具允许计划人员估算成本和工作量,还可以对重要的项目变量(例如,交付日期或人员配置)进行假设分析。尽管目前已有很多自动估算工具(参看本章后面的补充材料),但它们具有相同的本质特性,都能实现以下 6 项一般性功能 [Jon96]。

1. 估算项目可交付产品的规模——估算一个或多个软件工作产品的“规模”。工作产品包括软件的外部表示(如屏幕、报表)、软件本身(如 KLOC)、交付的功能(如功能点)及描述性信息(如文档)。
2. 选择项目活动——选择适当的过程框架,指定软件工程任务集。
3. 预测人员配置标准——指定可用的工作人员数量。由于可用人员和工作(预测的工作量)之间完全是非线性关系,因此这是一项重要输入。

4. 预测软件工作量——估算工具使用一个或多个估算模型(33.7 节)来预测软件工作量,这些模型能将交付的项目规模与开发所需的工作量联系起来。

5. 预测软件成本——如果给出第 4 步的结果,再分别指定第 2 步中确定的项目活动的劳动力价格,就可以计算出成本。

6. 预测软件进度计划——当工作量、人员配置和项目活动已知后,可以根据本章后面讨论的工作量分配推荐模型,来分配各个软件工程活动的人员,从而制定出进度计划草案。

当把不同的估算工具应用于相同的项目数据时,估算结果会有比较大的变动范围。更重要的是,有时候预测值会与实际值差异很大。这再次证明了应该把估算工具的输出看作是一个“数据点”,再从中导出估算,而不是将其作为估算的唯一来源。

33.7 经验估算模型

计算机软件估算模型使用由经验导出的公式来预测工作量，工作量是 LOC 或 FP 的函数^①。LOC 或 FP 的值采用 33.6.3 节和 33.6.4 节所描述的方法进行估算，但不使用该节中的表，而是将 LOC 或 FP 的结果值代入到估算模型中。

关键点 估算模型反映的是导出其所基于的项目集，因此模型具有领域敏感性。

用以支持大多数估算模型的经验数据都是从有限的项目样本中得出的。因此，还没有一种估算模型能够适用于所有软件类型和开发环境。所以，从这些模型中得到的结果应该慎重使用。

应该对估算模型进行调整，以反映当前项目的情况。应该使用从已完成项目中收集的数据对该模型进行检验——方法是将数据代入到模型中，然后将实际结果与预测结果进行比较。如果两者一致性很差，则在使用该模型前，必须对其进行调整和再次检验。

743

33.7.1 估算模型的结构

典型的估算模型是通过对以往软件项目中收集的数据进行回归分析而导出的。这种模型的总体结构表现为下面的形式 [Mat94]:

$$E = A + B \times (e_v) C \quad (33.3)$$

其中， A 、 B 、 C 是经验常数， E 是工作量（以人月为单位）， e_v 是估算变量（LOC 或 FP）。除了式 (33.3) 所表示的关系外，大多数估算模型都有某种形式的项目调整成分，使得 E 能够根据其他的项目特性（例如，问题的复杂性、开发人员的经验、开发环境）加以调整。从任何一个根据经验得出的模型可以看出，必须根据项目特定环境的要求对估算模型进行调整。

33.7.2 COCOMO II 模型

Barry Boehm [Boe81] 在其关于“软件工程经济学”的经典著作中介绍了一种层次结构的软件估算模型，称为 COCOMO (COConstructive COst MOdel, 构件性成本模型)。最初的 COCOMO 模型是得到产业界最广泛使用和讨论的软件成本估算模型之一。现在，它已经演化成更全面的估算模型，称为 COCOMOII [Boe00]。与其前身一样，COCOMOII 实际上也是一种层次结构的估算模型，应用于软件过程的不同“阶段”。

COCOMO II 模型与所有软件估算模型一样，也需要使用规模估算信息，在模型层次结构中有三种不同的规模估算选择：对象点^②、功能点和源代码行。

33.7.3 软件方程

软件方程 [Put92] 是一个动态的多变量模型，它假定在软件开发项目的整个生命周期中有特定的工作量分布。该模型是根据从 4000 多个当代软件项目中收集的生产率数据导出的。根据这些数据，我们导出以下形式的估算模型：

① 33.6.7 节给出了使用用例作为独立变量的经验模型。但是，到目前为止，在文献中出现的非常少。

② 对象点是一种间接的软件测量。计算对象点时，使用如下的计数值：(1) (用户界面的) 屏幕数；(2) 报表数；(3) 构造应用时可能需要的构件数，加上复杂度系数。

$$E = \frac{LOC \times B^{0.33}}{P^3} \times \frac{1}{t^4} \quad (33.4)$$

744 其中:

E 为工作量, 以人月或人年为单位。

t 为项目持续时间, 以月或年为单位。

B 为“特殊技能因子”^①。

P 为“生产率参数”, 它反映了: 总体的过程成熟度及管理实践; 采用良好的软件工程实践的程度; 使用的程序设计语言的水平; 软件环境的状态; 软件团队的技能和经验; 应用的复杂性。

对于实时嵌入式软件的开发, 典型值是 $P = 2000$; 对于电信及系统软件, $P = 10000$; 对于商业系统应用, $P = 28000$ 。当前情况下的生产率参数可以根据以往开发工作中收集到的历史数据来导出。

应该注意到, 软件方程有两个独立的参数: (1) 规模的估算值 (以 LOC 为单位); (2) 项目持续时间, 以月或年为单位。

Putnam 和 Myers[Put92] 为了简化估算过程, 并将估算模型表示成更通用的形式, 他们给出了一组从软件方程中导出来的方程式。最短开发时间定义为:

$$t_{\min} = 8.14 \frac{LOC}{P^{0.43}}, \text{ 以月为单位, 用于 } t_{\min} > 6 \text{ 个月的情况} \quad (33.5a)$$

$$E = 180Bt^3, \text{ 以人月为单位, 用于 } E \geq 20 \text{ 人月的情况} \quad (33.5b)$$

注意方程式 (33.5b) 中的 t 是以年为单位。

对本章前面所讨论的 CAD 软件, 使用方程式 (33.5), 令 $P = 12000$ (对科学计算软件的推荐值):

$$t_{\min} = 8.14 \times \frac{33200}{12000^{0.43}} = 12.6 \text{ (月)}$$

$$E = 180 \times 0.28 \times 1.05^3 = 58 \text{ (人月)}$$

由软件方程得到的结果与 33.6 节产生的估算值非常一致。同 33.7.2 节中提到的 COCOMO 模型一样, 软件方程将继续演化下去, 关于该估算方法扩展版本的进一步讨论参见 [Put97b]。

745

33.8 面向对象项目的估算

使用明确为面向对象软件设计的估算技术来对软件成本估算的传统方法进行补充, 这种做法是值得的。Lorenz 和 Kidd[Lor94] 给出了下列方法:

1. 使用工作量分解、FP 分析和任何其他适用于传统应用的方法进行估算。
2. 使用需求模型 (第 10 章) 建立用例并确定用例数。要认识到随着项目的进展, 用例数可能会改变。
3. 由需求模型确定关键类 (在第 10 章中称为分析类) 的数量。
4. 对应用的界面类型进行归类, 以确定支持类的乘数。对应于没有图形用户界面、基于文本的用户界面、传统的图形用户界面、复杂的图形用户界面这 4 种界面类型, 相应

① 随着“对集成、测试、质量保证、文档和管理技能的需求的增长”, B 的值缓慢增加 [Put92]。对于较小的程序 ($KLOC=5 \sim 15$), $B=0.16$ 。对于超过 70KLOC 的程序, $B=0.39$ 。

的支持类乘数分别是 2.0、2.25、2.5 和 3.0。关键类的数量（第 3 步）乘上乘数就得到了支持类数量的估算值。

5. 将类的总数（关键类 + 支持类）乘以每个类的平均工作单元数。Lorenz 和 Kidd 建议每个类的平均工作单元数是 15 ~ 20 人日。
6. 将用例数乘以每个用例的平均工作单元数，对基于类的估算做交叉检查。

33.9 特殊的估算技术

33.6 节 ~ 33.8 节讨论的估算技术可以用于任何软件项目。但是，当软件团队遇到一个持续时间非常短（以周计而不是以月计）的项目，中间又可能出现连续不断的变更时，通常应该对项目计划特别是估算进行简化^①。在下面的两小节中，研究两种特殊的估算技术。

33.9.1 敏捷开发的估算

由于敏捷项目（第 5 章）的需求是通过一组用户场景（如极限编程中的“故事”）来定义的，所以在项目计划阶段为每个软件增量开发一个非正式的、比较严谨并有意义的估算方法是可能的。敏捷项目的估算采用分解法，包括下列步骤。

1. 从估算目的出发，分别考虑每个用户场景（由最终用户或其他利益相关者在项目初期建立，等价于一个微型用例）。
2. 将场景分解成一组开发它所需要完成的软件工程任务。
- 3a. 分别估算每一项任务。注意，可以根据历史数据、经验模型或“经验”进行估算。
- 3b. 或者可以利用 LOC、FP 或其他某种面向规模的测量（如用例点）来估算场景的“规模”。
- 4a. 对每项任务的估算结果求和，就得到了对整个场景的估算值。
- 4b. 或者使用历史数据，将场景规模的估算值转换成工作量。
5. 将实现给定软件增量的所有场景的工作量估算值求和，就得到了该增量的工作量估算。

提问 当采用敏捷过程时，如何进行估算？

关键点 在敏捷项目的估算中，“规模”是指使用 LOC 或 FP 对用户场景总规模的估算。

由于软件增量开发所需的项目时间非常短（一般是 3 ~ 6 周），所以该估算方法用于两个目的：（1）确保增量中包含的场景数与可用资源相匹配；（2）在开发增量时，为工作量的分配提供依据。

33.9.2 WebApp 项目的估算

WebApp 项目常常采用敏捷过程模型。利用修改过的功能点测量，配合 33.9.1 节中简述的步骤，就可以进行 WebApp 的估算了。当把功能点用于 WebApp 的估算时，Roetzheim[Roe00] 给出了下列方法。

- 输入：每个输入屏幕或表单（例如，CGI 或 Java）、每个维护屏幕、每个标签页（无论你在什么地方使用带有标签页的编辑框时）。
- 输出：每个静态 Web 页、每个动态 Web 页脚本（例如，ASP、ISAPI 或其他 DHTML 脚本）和每个报表（无论是基于 Web 的还是管理的）。

① “简化”并不意味着消除。工期再短的项目都必须要做计划，而估算是可靠计划的基础。

- 表：数据库中的每个逻辑表，如果使用 XML 来存储文件中的数据，则指的是每一个 XML 对象（或 XML 属性集）。
- 界面：定义为逻辑文件（例如，唯一记录格式），作为系统与外部的边界。
- 查询：每一个查询都是对外发布的界面，或者使用面向消息的界面。典型的例子如 DCOM 或 COM 的外部引用。

对于 WebApp 而言，功能点（按照上述的说明）是一个合理的规模指标。

软件工具 工作量和成本估算

[目标] 工作量和成本估算工具从项目具有的特性和构建项目的环境出发，为项目团队提供项目所需工作量、持续时间和成本的估算。

[机制] 通常，成本估算工具要利用从本地项目中导出的历史数据库、整个产业中收集的数据、以及用于导出工作量、项目持续时间和成本估算的经验模型（例如，COCOMO II）。这些工具以项目特性和开发环境作为输入，能够给出估算结果的变动范围。

[代表性工具]^①

- Costar。由 Softstar Systems (www.softstarsystems.com) 开发，使用 COCOMO II 模型进行软件估算。
- Cost Xpert。由 Cost Xpert Group (www.costxpert.com) 开发，集成了多种估算模型和一个历史项目数据库。
- Construx Professional。由 Construx ([http://](http://www.construx.com)

www.construx.com/Resources/Construx_Estimate/) 开发，基于 Putnam 模型和 COCOMO II。

- Knowledge Plan。由 Software Productivity Research (www.spr.com) 开发，是一个完整的估算软件包，使用功能点作为主要的输入数据。
- Price S。由 Price Systems (www.pricesystems.com) 开发，是最古老的、使用最广泛的估算工具之一，用于大型软件开发项目。
- SEER/SEM。由 Galorath (www.galorath.com) 开发，提供了全面的估算能力、敏感分析、风险评估和其他特性。
- SLIM-Estimate。由 QSM (www.qsm.com) 开发，利用完善的“产业知识库”对使用本地数据导出的估算进行“完整性检查”。

33.10 自行开发或购买的决策

在许多软件应用领域中，直接获取（购买）计算机软件常常比自行开发的成本要低得多。软件工程管理者面临着要做出自行开发还是购买的决策问题，而且由于存在多种可选获取方案而使得决策更加复杂：（1）购买成品构件（或取得使用许可）；（2）购买“具有完全经验”或“具有部分经验”的软件构件（见 33.4.2 节），并进行修改和集成，以满足特定的需求；（3）由外面的承包商根据买方的规格说明定制开发。

根据要购买软件的迫切程度及最终成本来确定软件获取的步骤。在有些情况下（例如，

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

低成本的应用), 购买并试用方式比购买软件包并对其做冗长评估的成本要低。在最后的分析中, 自行开发或购买的决策是根据以下条件决定的: (1) 软件产品的交付日期是否比内部开发要快? (2) 购买的成本加上定制的成本是否比内部开发该软件的成本低? (3) 外部支持 (例如, 维护合同) 的成本是否比内部支持的成本低? 这些条件可以应用于上述每种可选的获取方案中。

33.10.1 创建决策树

可以使用统计技术对上述步骤进行扩充, 如决策树分析[⊖]。图 33-5 描述了一个基于软件的系统 X 的决策树。在这个例子中, 软件工程组织能够: (1) 从头开始构建系统 X; (2) 复用现有的“具有部分经验”的构件来构建系统; (3) 购买现成的软件产品并进行修改, 以满足当前项目的需要; (4) 将软件开发承包给外面的开发商。

提问 在对自行开发还是购买进行决策时, 是否有系统化的方法对决策相关的选项进行排序?

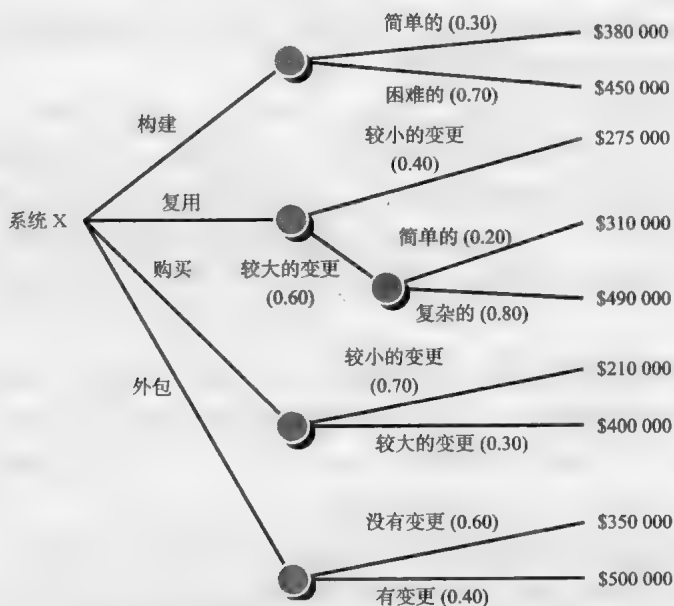


图 33-5 一个支持自行开发 / 购买决策的决策树

如果从头开始构建系统, 那么这项工作难度较大的概率是 70%。项目计划人员使用本章前面讨论的估算技术可估计出, 一项难度较大的开发工作将需要 450000 美元的成本, 而一项“简单的”开发工作估计需要 380000 美元。沿决策树的任一分支进行计算, 得到成本的预期值是 (K 表示千美元):

$$\text{预期成本} = \sum (\text{路径概率})_i \times (\text{估算的路径成本})_i$$

其中, i 是决策树的某条路径。对于“构建系统”这条路径而言:

$$\text{预期成本}_{\text{构建}} = 0.30 \times 380K + 0.70 \times 450K = 429K$$

沿着决策树的其他路径, 分别给出了多种情况下, “复用”“购买”和“外包”方式的项目成本。这些路径的预期成本分别是 (K 表示千美元):

$$\text{预期成本}_{\text{复用}} = 0.40 \times 275K + 0.60 \times (0.20 \times 310K + 0.80 \times 490K) = 382K$$

⊖ 在 http://en.wikipedia.org/wiki/Decision_tree 上能找到关于决策树分析的有价值的介绍。

$$\text{预期成本}_{\text{购买}} = 0.70 \times 210\text{K} + 0.30 \times 400\text{K} = 267\text{K}$$

$$\text{预期成本}_{\text{外包}} = 0.60 \times 350\text{K} + 0.40 \times 500\text{K} = 410\text{K}$$

根据图 33-5 给出的路径概率及估算成本，可以看出“购买”方式具有最低的预期成本。

不过，应该注意到，在决策过程中还有许多准则（而不仅仅是成本）必须加以考虑。在最终决定使用构建、复用、购买或外包方式时，可用性、开发者/厂家/承包商的经验、与需求的一致性、本地的政策环境及变更的可能性等，这些都可能是影响判断的准则，当然这些也仅仅是其中一部分准则而已。

33.10.2 外包

每一个开发计算机软件的公司迟早都会问到一個基本问题：是否有什么方法能使我们以较低的价格获得所需的软件和系统？这个问题的答案不是唯一的，但对于这个问题的情绪化的回答经常是一句话：外包。

在概念上，外包是非常简单的。软件工程活动被承包给第三方，他们能以较低的成本并有希望以较高的质量来完成这项工作。公司内部需要做的软件工作已经降至仅仅是合同管理活动^①。

做外包决策时要从战略上或战术上考虑。在战略层上，业务管理人员要考虑大部分软件工作是否可以承包给其他厂商。在战术层上，项目经理要确定通过外包部分软件工作是否能够最好地完成项目的部分或全部。

引述 与内部开发相比，外包需要更高超的管理，这是一条规律。

Steve McConnell

[750]

不考虑太多其他因素，外包的决策常常是财务的决策。关于外包财务分析的详细探讨超出了本书的范围，其他书（如 [Min95]）中有很好的讨论。不过，从正反两方面考察一下外包的决策是值得的。

从正面来看，由于减少了软件人员及相应设备（例如，计算机、基础设施），因此外包通常能够节约成本。从反面来看，公司失去了对其所需软件的部分控制权。这是因为软件是一种技术，因而不同于公司的系统、服务和产品。这样，公司就会冒着将其竞争命运交到第三方手中的风险。

毫无疑问，向外包发展的趋势将会继续，减缓这种趋势的唯一方法是让人们认识到所有层次上的软件工作都具有强烈的竞争。生存的唯一办法就是具有与外包厂商同等的竞争力。

SafeHome 外包

[场景] 项目初期，CPI 公司的会议室。

[人物] Mal Golden，产品开发高级经理；Lee Warren，工程经理；Joe Camalleri，业务开发副主管；Doug Miller，软件工程项目经理。

[对话]

Joe: 我们正在考虑将产品的 SafeHome 软件工程部分外包出去。

Doug (很震惊): 这是什么时候的事？

Lee: 我们得到了一个国外开发者的报价，比你们小组认为要花费的成本低 30%。在这儿（将报价交给 Doug 看）。

Mal: 你知道，Doug，我们正在设法降低成本，30%，是 30% 啊。此外，这些人非常受推崇。

Doug (深深地吸一口气并努力保持镇静):

^① 在广义上，可以将外包看作是一项从软件工程组织外获取软件或软件构件的活动。

你们让我很惊讶。不过在最后决定之前，先听听几条意见吧。

Joe (点头): 当然，你说吧。

Doug: 以前，我们还没有同这家外包公司合作过，对吗？

Mal: 对，但是……

Doug: 并且他们提到，对规格说明的任何变更都要追加另外的费用，对吗？

Joe (皱着眉): 是真的，不过我们预期它会相当稳定。

Doug: 一个错误的假定，Joe。

Joe: 唔……

Doug: 不出几年，我们就可能发布该产品的新版本。我们有理由假定软件将提供很多新特性，对吗？

(都点头)

Doug: 我们以前曾经协调过国际项目吗？

Lee (担忧地看着): 没有，但是我得知……

Doug (设法抑制着愤怒): 你要告诉我的就是：(1) 我们将要同一个不了解的厂商合作。(2) 这项工作的成本并不像他们认为

的那样低。(3) 事实上，无论他们第一次做成什么样，在多次产品发布中，我们都必须与他们合作。(4) 我们将要在职学习关于国际项目的相关知识。

(都保持沉默)

Doug: 我认为这是错误的。我希望你们用一天时间重新考虑。如果在内部完成这项工作，我们将有更多的控制权。我们拥有专门的技术，并且我能担保不会花费更多，而且风险更低。我知道，和我一样，你们都是反对风险的。

Joe (皱着眉): 你已经指出了几个优点，但是你具有该项目在内部开发的既得利益。

Doug: 那是真的，但是它不能改变事实。

Joe (叹气): 好吧，先把这个问题搁置一两天，好好考虑一下，然后再开会做最后决定。Doug，我可以和你私下谈谈吗？

Doug: 当然，我实在是想确保我们做的事情一切顺利。

751

33.11 小结

在项目开始之前，软件项目计划人员必须先估算三件事：花费多长时间，需要多少工作量，涉及多少人员。此外，计划人员还必须预测所需要的资源（硬件和软件）及蕴涵的风险。

范围陈述能够帮助计划人员使用一种或多种技术进行估算，这些技术主要分为两大类：分解和经验建模。分解技术需要划分出软件的主要功能，接着估算：(1) LOC 的数量；(2) 信息域内的选择值；(3) 用例的数量；(4) 实现每个功能所需的人月数；(5) 每个软件工程项目活动所需的人月数。经验技术使用根据经验导出的关于工作量和时间的公式来预测这些项目数值。可以使用自动工具来实现特定的经验模型。

对项目做精确估算时，一般至少要用到上述三种技术中的两种。通过对不同技术产生的估算值进行比较和调和，计划人员更有可能得到精确的估算。软件项目估算永远不会是一门精确的科学，但是，把可靠的历史数据与系统化的技术结合起来能够提高估算的精确度。

习题与思考题

- 33.1 假设你是一家开发家用机器人软件公司的项目经理，你已经承接了为草坪割草机器人开发软件的项目。写一个范围陈述来描述该软件，确定你的范围陈述是“界定的”。如果你对机器人不熟悉，在你开始写作之前先做一些调研工作。还要说明你对所需硬件的设想。或者，你也可以选

择其他感兴趣的问题，而不做草坪割草机器人。

- 33.2 在 33.1 节简要讨论了软件项目的复杂性。列出影响项目复杂性的软件特性（例如，并发操作、图形输出），按其对项目的影响程度顺次排列。
- 33.3 在计划过程中，性能是一个重要的考虑因素。针对不同的软件应用领域，分别讨论如何以不同的方式来解释性能。
- 33.4 对你在问题 33.1 中描述的机器人软件进行功能分解。估算每个功能的规模（用 LOC）。假定你所在组织的平均生产率是 450LOC/pm，劳动力价格是每人月 7000 美元，使用本章所讲的基于 LOC 的估算技术来估算构建该软件所需的工作量及成本。
- 33.5 使用“软件方程”来估算草坪割草机器人软件。假设采用方程式（33.4），且 $P=8000$ 。
- 33.6 建立一个电子表格模型，实现本章所述的一种或多种估算技术。或者从基于 Web 的资源中获取一个或多个在线估算模型。
- 33.7 组建一个项目团队，开发软件工具来实现本章所介绍的每种估算技术。
- 33.8 有一点似乎很奇怪：成本和进度估算是在软件项目计划期间完成的——在详细的软件需求分析或设计之前进行。你认为为什么会这样？是否存在不需要这样做的情况？

扩展阅读与信息资源

大多数软件项目管理书籍都包含了对项目估算的讨论。项目管理研究所（《PMBOK Guide》，PMI，2001）、Wysoki（《Effective Project Management：Traditional, Agile, Extreme》，6th ed., Wiley, 2011）、Lewis（《Project Planning Scheduling and Control》，5th ed., McGraw-Hill, 2010）、Kerzner（《Project Management: A Systems Approach to Planning, Scheduling, and Controlling》，10th ed., Wiley, 2009）、Bennatan（《On Time, Within Budget：Software Project Management Practices and Techniques》，3rd ed., Wiley, 2000）以及 Phillips[Phi98] 都提供了有用的估算指南。

McConnell（《Software Estimation：Demystifying the Black Art》，Microsoft Press, 2006）撰写了实用指南，为那些必须估算软件成本的人提供了有价值的指导。Parthasarathy（《Practical Software Estimation》，Addison-Wesley, 2007）强调以功能点作为估算度量。Hill（《Practical Software Project Estimation》，McGraw-Hill Osborne Media, 2010）以及 Laird 和 Brennan（《Software Measurement and Estimation：A Practical Approach》，Wiley-IEEE Computer Society Press, 2006）论述了测量及其在软件估算中的应用。Pfleeger（《Software Cost Estimation and Sizing Methods, Issues and Guidelines》，RAND Corporation, 2005）给出了一个浓缩的指南，描述了估算的很多基本原理。Jones（《Estimating Software Costs》，2nd ed., McGraw-Hill, 2007）撰写的著作是对模型和数据最全面的论述之一，这些模型和数据可用于各个应用领域的软件估算。Coombs（《IT Project Estimation》，Cambridge University Press, 2002）以及 Roetzheim 和 Beasley（《Software Project Cost and Schedule Estimating：Best Practices》，Prentice Hall, 1997）介绍了很多有用的模型，并逐步给出了生成最可能估算的指南。

大量的关于软件估算的信息可以在网上获得。最新的参考文献参见 SEPA 网站 www.mhhe.com/pressman 下的“software engineering resources”。

项目进度安排

要点浏览

概念: 你已经选择了合适的过程模型, 确定了必须完成的软件工程任务, 估算了工作量和人员数量, 明确了项目最后期限, 甚至已经考虑了风险, 现在是综合运用它们的时候了。也就是说, 你应该创建一个软件工程任务网络, 该网络将使你能够按时完成工作。任务网络创建完成之后, 你必须为每一个任务确定责任, 还要确保完成这些责任, 并在风险到来时调整该网络。简单地说, 这就是软件项目进度安排和跟踪。

人员: 在项目级, 是那些使用从软件工程师处获得的信息的软件项目管理者们。在个体级, 是软件工程师自己。

重要性: 为了建造复杂的系统, 很多软件工程任务会并行地进行, 而且在一个任务中得到的工作结果可能对在另一个任务中将要进行的工作具有深远的影响。

如果没有进度安排, 任务之间的这种相互依赖性将是非常难以理解的。实际上, 没有一个详细的进度安排, 要评估中等程度或大型的软件项目的进展情况也是不可能的。

步骤: 软件过程模型中规定的软件工程任务要根据具体实现的功能进行细化; 为每一个任务分配工作量和工期; 创建任务网络 (也称为“活动网络”), 使得软件团队能够在最后期限之前完成项目。

工作产品: 项目进度安排和相关的信息。

质量保证措施: 正确的进度安排要求: (1) 网络中包含所有的任务; (2) 给每个任务合理分配工作量和时间; (3) 明确指出任务间的依赖关系; (4) 资源应分配给具体要完成的工作; (5) 提供短时间段间隔的里程碑, 以便于过程跟踪。

20 世纪 60 年代后期, 一位热情的青年工程师受命为一个自动制造业应用项目“编写”计算机程序。选择他的原因非常简单, 因为在整个技术小组中他是唯一参加过计算机编程培训班的人。这位工程师对汇编语言的 IN 和 OUT 指令以及 Fortran 语言有所了解, 但是却根本不懂软件工程, 更不用说项目进度安排和跟踪了。

他的老板给了他一大堆相关的手册, 口头描述了需要做些什么。年轻人被告知该项目必须在两个月之内完成。

他阅读了这些手册, 想好了解决方法, 就开始编写代码。两周之后, 老板将他叫到办公室询问项目进展情况。

“非常顺利,” 工程师以年轻人的热情回答道, “这个项目远比我想象的简单, 我差不多已经完成了 75% 的任务。”

关键概念

- 关键路径
- 净值
- 工作量分配
- 人员与工作量
- 进度安排
- 原则
- Web APP 和移动项目
- 任务网络
- 时间盒
- 时序图
- 跟踪
- 工作分解

老板笑了，然后鼓励这个青年工程师继续努力工作，准备好一周后再汇报工作进度。

一周之后老板将年轻人叫到办公室，问道：“现在进度如何？”

“一切顺利，”年轻人回答说，“但是我遇到了一些小麻烦。我会排除这些困难，很快就可以回到正轨上来。”

“你觉得在最后期限之前能完成吗？”老板问道。

“没问题，”工程师答道，“我差不多已经完成 90% 了。”

如果你在软件领域中工作过几年，你一定可以将这个故事写完。毫不奇怪，青年工程师在整个项目工期内始终停留在 90% 的进度上，实际上直到交付期限之后一个月（在别人的帮助下）才完成。^①

在过去的 50 年间，这样的故事在不同的软件开发者中已经重复了成千上万次，这是为什么呢？

34.1 基本概念

虽然软件延期交付的原因很多，但是大多数都可以追溯到下面列出的一个或多个根本原因上：

- 不切实际的项目最后期限，由软件团队以外的某个人制定，并强加给软件团队的管理者和开发者。
- 客户需求发生变更，而这种变更没有在项目变更进度表上预先安排。
- 对完成该工作所需的工作量和资源数量估计不足。
- 在项目开始时，没有考虑到可预测的和不可预测的风险。
- 出现了事先无法预计的技术难题。
- 出现了事先无法预计的人力问题。
- 由于项目团队成员之间的交流不畅而导致的延期。
- 项目管理者未能发现项目进度拖后，也未能采取措施来解决这一问题。

在软件行业中，人们对过于乐观的（即“不切实际的”）项目最后期限已经司空见惯。从设定项目最后期限的人的角度来看，有时候这样的项目最后期限是合理的。但是常识告诉我们，合理与否还必须由完成工作的人员来判断。

第 33 章讨论的估算方法和本章中的进度安排技术，通常都需要在规定的最后期限约束下进行。如果最乐观的估算都表明该项目最后期限是不现实的，一个称职的项目管理者就应该“保护其团队免受不适当的（进度安排）压力……（并）将这种压力反映给施加压力的一方”[Pag85]。

举例说明，假定你负责的软件团队受命开发一个医疗诊断仪器的实时控制器，该控制器要在 9 个月之内推向市场。在你进行了仔细的估算和风险分析（第 35 章）之后得到的结论是：在现有人员条件下，需要 14 个月的时间才能完成这一软件。你下一步该怎么办呢？

闯进客户的办公室（这里的客户很可能是市场 / 营销人员）并要求修改交付日期似乎不太现实。外部市场压力已经决定了交付日期，届时必须发布产品。而（从事业前途的角度出

引述 在所有的软件项目中，过于理性或缺少理性的进度安排可能最具破坏性影响。

Capers Jones

引述 我热爱最后期限，我喜欢它们飞过时所发出的声音。

Douglas Adams

^① 你可能觉得惊奇，但这个故事是我自己经历过的事（RSP）。

发)拒绝这一项目同样是鲁莽的。那么应该怎么办呢?在这种情况下,建议按照以下步骤进行处理:

1. 按照以往项目的历史数据进行详细的估算,确定项目的估算工作量和工期。
2. 采用增量过程模型(第4章)制定软件工程策略,以保证能够在规定的交付日期提供主要功能,而将其他功能的实现推到以后。然后将这一计划做成文档。
3. 与客户交流,并(用详细的估算结果)说明为什么规定的交付日期是不现实的。一定要指出所有这些估算都是基于以往的项目实践,而且为了在目前规定的交付期限完成该项目,与以往相比在工作效率上必须提高的百分比^①。可以做如下解释:

我认为在XYZ控制器软件的交付日期方面存在问题,我已经将一份以往项目中生产率的简化明细分类表和以多种不同方式进行的项目估算提交给各位,你们会注意到,在假设与以往的生产率相比有20%的提高的情况下,交付时间仍然需要14个月而不是9个月。

4. 将增量开发策略作为可选计划提交给客户:

我们有几种方案,我希望各位能够在这些方案中做出决策。第一个方案,我们可以增加预算,并引入额外的资源,以使我们能够在9个月内完成这项工作。但是应该知道由于时间限制过于苛刻,这样做将会增加质量变差的风险^②;第二个方案,去掉一部分需求中所列出的软件功能和特性,由此得到功能稍弱的产品的最初版本,但是我们可以对外宣布全部功能,并在总共14个月的时间内交付这些功能;第三个方案,是不顾现实条件的约束,而希望项目能够在9个月时间内完成,结果是我们竭尽全力,但是却无法向客户提供任何功能。我希望你们会赞成我的观点,第三个方案是不可行的。过去的历史和我们最乐观的估算都表明这是不现实的,是在制造一场灾难。

尽管这样做会有人抱怨,但如果你给出了基于准确历史数据的可靠估算,那么最终的谈判结果将可能是选择方案1或方案2。不现实的交付期限就不存在了。

34.2 项目进度安排概述

曾经有人向Fred Brooks请教软件项目的进度是怎样被延误的?他的回答既简单又深刻:“某天某时。”

技术性项目(不论它是涉及水力发电厂建设,还是操作系统开发)的现实情况是:在实现大目标之前必须完成数以百计的小任务。这些任务中有些是处于主流之外的,其进度不会影响到整个项目的完成日期。而有些任务却是位于“关键路径”上,如果这些“关键”任务的进度拖后,则整个项目的完成日期就会受到威胁。

项目管理者职责是确定所有的项目任务,建立相应的网络来描述它们之间的依赖关系,明确网络中的关键任务,然后跟踪关键任务的进展,以确保能够在“某天某时”发现进度延误情况。为了做到这一点,管理者

提问 当管理者要求的项目最后期限无法实现时,应该怎么办?

756

建议 为达到项目管理者的目标所必须完成的任务不应该以手工方式来安排,有很多优秀的项目进度安排工具可供使用。

① 如果生产率提高10%~25%,实际上是有完成这个项目的。但大多数情况是,所需提高的团队生产率要高于50%。所以说这是不切实际的期望。

② 你还可以补充说:“人员数量的增加不会成比例地缩短完成该项目所需要的时间。”

必须建立相当详细的进度表,使得项目管理者能够监督进度,并控制整个项目。

软件项目进度安排 (software project scheduling) 是一种活动,它通过将工作量分配给特定的软件工程任务,从而将所估算的工作量分配到计划的项目工期内。但要注意的是,进度是随时间而不断演化的。在项目计划早期,建立的是一张宏观进度表,该进度表标识了所有主要的过程框架活动和这些活动所影响的产品功能。随着项目的进展,宏观进度表中的每个条目都会被细化成详细的进度表,这样就标识了特定的(完成一个活动所必须实现的)软件活动和任务,同时也进行了进度安排。

可以从两个不同的角度来讨论软件工程项目的进度安排。第一种情况,计算机系统的最终发布日期已经确定(而且不能更改),软件开发组织必须将工作量分布在预先确定的时间框架内。第二种情况,假定已知大致的时间界限,但是最终发布日期由软件工程开发组织自行确定,工作量是以能够最好地利用资源的方式来进行分配,而且在对软件进行仔细分析之后才决定最终发布日期。但不幸的是,第一种情况发生的频率远远高于第二种情况。

引述 过于乐观的进度安排并不会缩短实际进度,反而会拖后进度。
Steve McConnell

34.2.1 基本原则

就像软件工程的所有其他领域一样,软件项目进度安排也有很多的 basic 指导原则:

划分 (compartmentalization)。必须将项目划分成多个可以管理的活动和任务。为了实现项目的划分,产品和过程都需要进行分解。

相互依赖性 (interdependency)。划分后的各个活动或任务之间的相互依赖关系必须是明确的。有些任务必须按顺序出现,而有些任务则可以并发进行。有些活动只有在其他活动产生的工作产品完成后才能够开始,而有些则可以独立进行。

时间分配 (time allocation)。每个要进行进度安排的任务必须分配一定数量的工作单位(例如,若干人日的工作量)。此外,还必须为每个任务指定开始日期和完成日期,任务的开始日期和完成日期取决于任务之间的相互依赖性以及工作方式是全职还是兼职。

工作量确认 (effort validation)。每个项目都有预定人员数量的软件团队参与。在进行时间分配时,项目管理者必须确保在任意时段中分配的人员数量不会超过项目团队中的总人员数量。例如,某项目分配了3名软件工程师(例如,每天可分配的工作量为3人日^①)。在某一天中,需要完成7项并发的任务,每个任务需要0.5人日的工作量,在这种情况下,所分配的工作量就大于可供分配的工作量。

确定责任 (defined responsibility)。进度计划安排的每个任务都应该指定特定的团队成员来负责。

明确输出结果 (defined outcome)。进度计划安排的每个任务都应该有一个明确的输出结果。对于软件项目而言,输出结果通常是一个工作产品(例如,一个模块的设计)或某个工作产品的一部分。通常可将多个工作产品组合成可交付产品。

确定里程碑 (defined milestone)。每个任务或任务组都应该与一个项目里程碑相关联。当

关键点 在进度安排时,要划分工作,描述任务间的相互依赖性,为每个任务分配工作量和时间,确定责任、输出结果和里程碑。

^① 实际上,由于与工作无关的会议、病假、休假以及各种其他原因,可供分配的工作量要少于3人日。但在这里,我们假定员工时间是100%可用的。

一个或多个工作产品经过质量评审(第19章)并且得到认可时,就标志着一个里程碑的完成。随着项目进度的推进,会应用到上述的每一条原则。

758

34.2.2 人员与工作量之间的关系

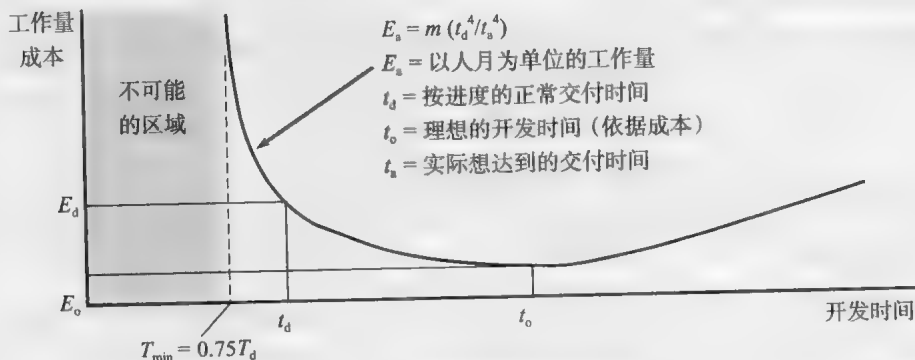


图 34-1 工作量和交付时间的关系

许多负责软件开发工作的管理者仍然普遍坚信这样一个神话:“即使进度拖后,我们也总是可以在项目后期增加更多的程序员来跟上进度。”不幸的是,在项目后期增加人手通常会对项目产生破坏性的影响,其结果是使进度进一步拖延。后期增加的人员必须学习这一系统,而培训他们的人员正是那些一直在工作着的人,当他们进行教学时,就不能完成任何工作,从而使项目进一步延后。

建议 如果必须给一个已经拖延的项目增加人员,就要确保已将详细划分的任务分配给他们。

除去学习系统所需的时间之外,整个项目中,新加入人员将会增加人员之间交流的路径数量和交流的复杂度。虽然交流对于一个成功的软件开发项目而言绝对是必不可少的,但是每增加一条新的交流路径就会增加额外的工作量,从而需要更多的时间。

多年以来的经验数据和理论分析都表明项目进度是具有弹性的。即在一定程度上可以缩短项目交付日期(通过增加额外资源),也可以拖延项目交付日期(通过减少资源数量)。

PNR(Putnam-Norden-Rayleigh)曲线^①表明了一个软件项目中所投入的工作量与交付时间的关系。项目工作量和交付时间的函数关系曲线如图34-1所示。图中的 t_0 表示项目最低交付成本所需的最少时间(即花费工作量最少的项目交付时间),而 t_0 左边(即当我们想提前交付时)的曲线是非线性上升的。

关键点 PNR曲线表明,拖延项目交付时间可以显著地降低项目成本。

举一个例子,假设一个软件项目团队根据进度安排和现有的人员配置,估算所需要的工作量应为 E_d ,正常的交付时间应为 t_d 。虽然可以提前交付,但曲线在 t_d 的左侧急剧上升。事实上,PNR曲线不仅说明了项目的交付时间不能少于 $0.75t_d$,如果想更少,项目会进入“不可能的区域”,并面临着很高的失败风险;还说明了最低成本的交付时间 t_0 应该满足 $t_0 = 2t_d$,即拖延项目交付可以明显降低成本,当然,这里的成本必须将与延期相关的营销成本排除在外。

759

在第33章中介绍的软件方程[Put92]就是来源于PNR曲线,它表明了完成一个项目的时间与投入该项目的人员工作量之间是高度非线性的关系。交付的代码(源程序代码)行数

① 相关的初始研究参见[Nor70]和[Put78]。

L 与工作量和开发时间的关系可以用下面的公式表示：

$$L = P \times E^{1/3} t^{4/3}$$

其中, E 是以人月为单位的开发工作量; P 是生产率参数, 它反映了影响高质量软件工程工作的各种因素的综合效果 (P 通常在 2000 到 12000 之间取值); t 是以月为单位的项目工期。

重新调整这个软件方程, 可以得到开发工作量 E 的计算公式:

$$E = \frac{L^3}{P^3 t^4} \quad (34.1)$$

其中, E 是在软件开发和维护的整个生命周期内所需的工作量 (按人年计算); t 是以年为单位的开发时间; 引入平均劳动力价格因素 (\$/人年) 之后, 开发工作量的计算公式还能够与开发成本相关联。

这一方程式引出了一些有趣的结果。假设有一个复杂的实时软件项目, 估计需要 33000 源代码行和 12 人年的工作量。如果项目团队有 8 个人, 那么项目大约需要 1.3 年的时间完成。但是如果将交付日期延长到 1.75 年, 则由式 (34.1) 所描述的模型所具有的高度非线性特性将得出以下结论:

$$E = \frac{L^3}{P^3 t^4} \approx 3.8 \text{ 人年}$$

这意味着通过将交付日期推迟 6 个月, 我们可以将项目团队的人数从 8 人减少到 4 人! 这一结果的有效性有待考证, 但是其含意却十分清楚: 通过在略为延长的时间内使用较少的人员, 可以实现同样的目标。

34.2.3 工作量分配

在第 33 章中讨论的各种软件项目估算技术最终都归结为对完成软件开发所需工作单位 (如人月) 的估算。软件过程中的工作量分配通常采用 40-20-40 法则。总体工作量的 40% 分配给前期的分析和设计, 40% 用于后期测试。因此, 你可以推断出编码工作 (20% 的工作量) 是次要的。

这种工作量分配方法只能作为指导原则^①。各个项目的特点决定了其工作量如何分配。用于项目计划的工作量很少超过 2% ~ 3%, 除非提交给组织的项目计划费用极高而且具有高风险。客户交流与需求分析大约占用 10% ~ 25% 的项目工作量, 用于分析或原型开发的工作量应该与项目规模和复杂度成正比地增长。通常有 20% ~ 25% 的工作量用于软件设计, 用于设计评审和随之而来的迭代开发也必须考虑。

因为在软件设计时投入了相当的工作量, 所以随后的编码工作就变得相对简单。总体工作量的 15% ~ 20% 就可以完成这一工作。测试和随之而来的调试工作将占用 30% ~ 40% 的软件开发工作量。软件的重要性决定了所需测试工作的分量, 如果软件系统是人命关天的 (即软件错误可能使人丧命), 就应该考虑分配更高的测试工作量比例。

建议 离项目的交付日期越来越近时, 你意识到, 不管参与工作的人数是多少, 工作均不能按计划完成。那就面对现实, 确定新的交付日期。

提醒 在软件过程的工作流程中应如何分配工作量?

① 现在, 40-20-40 法则不再适用。有些人认为用于分析和设计的工作量应超过总工作量的 40%。而相反的是, 敏捷开发的倡导者 (第 5 章) 认为 “前期” 工作应该越快越好, 团队应该快速进入构建阶段。

34.3 为软件项目定义任务集

无论选择哪一种过程模型，一个软件团队要完成的工作都是由任务集组成的，这些任务集使得软件团队能够定义、开发和最终维护计算机软件。没有能普遍适用于所有软件项目的任务集。适用于大型复杂系统的任务集可能对于相对简单的小型软件项目而言就过于复杂。因此，有效的软件过程应该定义一组任务集来满足不同类型项目的要求。

同第 3 章介绍的一样，任务集中包含了为完成某个特定项目所必须完成的所有软件工作任务、里程碑、工作产品以及质量保证过滤器。为了获得高质量的软件产品，任务集必须提供充分的规程要求，但同时又不能让项目团队负担不必要的工作。

在进行项目进度安排时，必须将任务集分布在项目时序图上。任务集应该根据软件团队所决定的项目类型和严格程度而有所不同。尽管很难建立一个全面详尽的软件项目分类方法，但是大多数软件组织遇到的项目一般属于下述类型：

1. 概念开发项目 (concept development project)。目的是为了探索某些新的业务概念或者某种新技术的应用。
2. 新应用开发 (new application development) 项目。根据特定的客户需求而承担的项目。
3. 应用增强 (application enhancement) 项目。对现有软件中最终用户可见的功能、性能或界面进行修改。
4. 应用维护项目 (application maintenance project)。以一种最终用户不会立即察觉到的方式对现有软件进行纠错、修改或者扩展。
5. 再工程项目 (reengineering project)。为了全部或部分重建一个现有 (遗留) 系统而承担的项目。

即使在单一的项目类型中，也会有许多因素影响任务集的选择。
[Pre05] 中描述了很多因素：项目的规模、潜在的用户数量、任务的关键性、应用程序的寿命、需求的稳定性、客户 / 开发者进行沟通的容易程度、可应用技术的成熟度、性能约束、嵌入式和非嵌入式特性、项目人员配置以及再工程因素等。综合考虑这些因素就形成了严格程度 (degree of rigor) 的指标，它将应用于所采用的软件过程中。

网络资源 适应性过程模型 (Adaptable Process Model, APM) 可以辅助不同的软件项目定义各自的任务集。有关 APM 的详细信息见 www.rspa.com/apm。

34.3.1 任务集举例

概念开发项目是在探索某些新技术是否可行时发起的。这种技术是否可行尚不可知，但是某个客户 (如营销人员) 相信其具有潜在的利益。概念开发项目的完成需要应用以下主要任务：

- 1.1 确定概念范围。确定项目的整体范围。
- 1.2 初步的概念策划。确定为承担项目范围所涵盖的工作组织应具有的工作能力。
- 1.3 技术风险评估。评估与项目范围中将要实现的技术相关联的风险。
- 1.4 概念证明。证明新技术在软件环境中的生命力。
- 1.5 概念实现。可以由客户方进行评审的方式实现概念的表达，而且在将概念推荐给其他客户或管理者时能够用于“营销”目的。
- 1.6 客户反应。向客户索取对新技术概念的反馈，并以特定的客户应用作为目标。

761

762

快速浏览以上任务，你应该不会有任何诧异。实际上概念开发项目的软件工作流程（以及其他所有类型的项目）与人们的常识相差无几。

34.3.2 主要任务的细化

上一节中所描述的主要任务（如软件工程活动）可以用来制定项目的宏观进度表。但是，必须将宏观进度表进行细化，以创建详细的项目进度表。细化工作始于将每项主要任务分解为一组子任务（以及相关的工作产品和里程碑）。

这里以“任务 1.1 确定概念范围”的任务分解为例。任务细化可以使用大纲格式，但是在这里将使用过程设计语言来说明“确定概念范围”这一活动的流程。

任务定义：任务 1.1 确定概念范围

1.1.1 确定需求、效益和潜在的客户

1.1.2 确定所希望的输出 / 控制和驱动应用程序的输入事件

开始任务 1.1.2

1.1.2.1 TR：评审需求的书面描述[⊖]

1.1.2.2 导出客户可见的输出 / 输入列表

1.1.2.3 TR：与客户一起评审输出 / 输入，并在需要时进行修改

结束任务 1.1.2

1.1.3 为每个主要功能定义功能 / 行为

开始任务 1.1.3

1.1.3.1 TR：评审在任务 1.1.2 中得到的输出和输入数据对象

1.1.3.2 导出功能 / 行为模型

1.1.3.3 TR：与客户一起评审功能 / 行为模型，并在需要时进行修改

结束任务 1.1.3

1.1.4 把需要在软件中实现的技术要素分离出来

1.1.5 研究现有软件的可用性

1.1.6 确定技术可行性

1.1.7 对系统规模进行快速估算

1.1.8 创建“范围定义”

结束任务 1.1 的任务定义

在过程设计语言中标注的这些任务和子任务共同构成了“确定概念范围”这个行动的详细进度表的基础。

34.4 定义任务网络

单个任务和子任务之间在顺序上存在相互依赖的关系。而且，当有多人参与软件工程项目时，多个开发活动和任务并行进行的可能性很大。在这种情况下，必须协调多个并发任务，以保证它们能够在后继任务需要其工作产品之前完成。

任务网络（task network）也称为活动网络（activity network），是项目任务流程的图形表示。有时将任务网络作为向自动项目进度安排工具中输

关键点 任务网络是描述任务间依赖关系和确定关键路径的有效机制。

⊖ TR 表示在此需要进行一次技术评审（第 20 章）。

入任务序列和依赖关系的机制。最简单的任务网络形式（创建宏观进度表时使用）只描述了主要的软件工程任务。概念开发项目的任务网络示意图如图 34-2 所示。

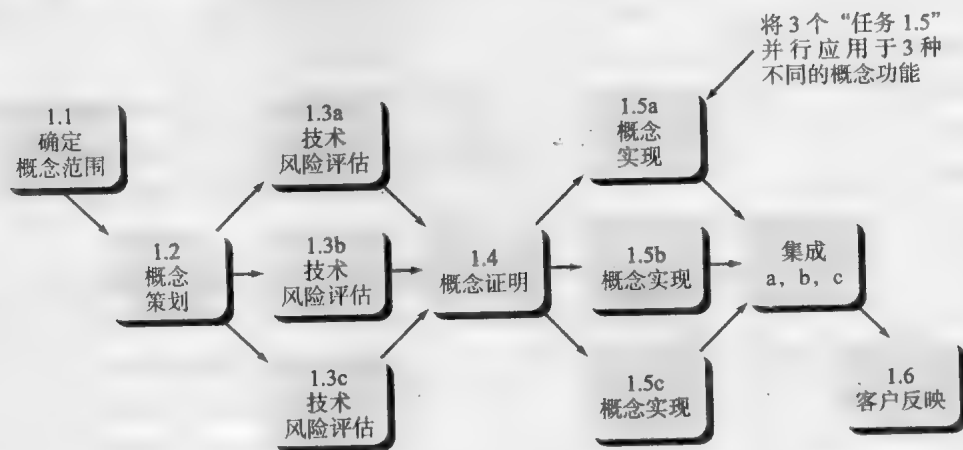


图 34-2 概念开发项目的任务网络

软件工程活动的并发本质导致了在进度安排上有很多要求。由于并行任务是异步发生的，所以项目管理者必须确定任务之间的依赖关系，以保证项目朝着最终完成的方向持续发展。另外，项目管理者应该注意那些位于关键路径（critical path）上的任务。也就是说，为了保证整个项目如期完成，必须保证这些任务能够如期完成。在本章的后面将详细讨论这些问题。

值得注意的是，图 34-2 中所示的任务网络是宏观的。详细的任务网络（详细进度表的前身）中应该对图 34-2 所示的各个活动加以扩展。例如，应该扩展任务 1.1，以表现 34.3.2 节所述的任务 1.1 细化中的所有任务。

34.5 进度安排

软件项目的进度安排与任何其他多任务工程工作的进度安排几乎没有差别。因此，通用的项目进度安排工具和技术不必做太多修改就可以应用于软件项目。

进度计划评估及评审技术（Program Evaluation and Review Technique, PERT）和关键路径方法（Critical Path Method, CPM）就是两种可以用于软件开发的项目进度安排方法。这两种技术都是由早期项目计划活动中已经产生的信息来驱动的，早期的项目计划活动包括：工作量的估算、产品功能的分解、适当过程模型和任务集的选择，以及所选择的任务的分解。

任务之间的依赖关系可以通过任务网络来确定。任务有时也称为项目的工作分解结构（Work Breakdown Structure, WBS），可以是针对整个产品，也可以是针对单个功能来进行定义。

PERT 和 CPM 两种方法都是定量划分的工具，可以使软件计划者完成：（1）确定关键路径——决定项目工期的任务链；（2）基于统计模型为单个任务进行“最有可能”的时间估算；（3）为特定任务的时间“窗口”计算“边界时间”。

引述 唯一要我们做出决定的就是怎样分配所给定的时间。

Gandalf,《指环王：护戒使者》

软件工具 项目进度安排

[目标] 项目进度安排工具的目标是使项目管理者能够确定工作任务, 建立工作任务之间的依赖关系, 为工作任务分配人员, 并能够形成各种图表, 以辅助对软件项目进行跟踪和控制。

[机制] 通常, 项目进度安排工具要求完成各任务的工作分解结构的规格说明, 或者创建任务网络。一旦确定了工作分解结构(大纲形式)或任务网络, 就可以为每一个任务指定开始和结束日期、分配人员、确定交付日期以及其他内容。然后, 项目进度安排工具可以生成多种时序图及其他表格, 使项目管理者能够对项目的任务流程进行评估。随着项目的进展, 这些信息可

以不断地进行更新。

[代表性工具]^①

- **AMS Realtime**。由 Advanced Management Systems (www.amsusa.com) 开发, 可以对各种规模和类型的项目做进度安排。
- **Microsoft Project**。由 Microsoft (www.microsoft.com) 开发, 是一个使用最广泛的 PC 项目进度安排工具。
- **4C**。由 4C Systems (www.4csys.com) 开发, 支持项目计划的各个方面, 包括进度安排。

项目管理软件厂商及其产品的详细清单见 www.infogoal.com/pmc/pmcswr.htm。

34.5.1 时序图

在创建软件项目进度表时, 计划者可以从一组任务(工作分解结构)入手。如果使用自动工具, 就可以采用任务网络或者任务大纲的形式输入工作分解结构, 然后再为每一项任务输入工作量、工期和开始日期。此外, 还可以将某些任务分配给特定的人员。

输入信息之后, 就可以生成时序图 (timeline chart), 也叫作甘特图 (Gantt chart)。可以为整个项目建立一个时序图, 也可以为各个项目功能或各个项目参与者分别建立各自的时序图。

图 34-3 给出了时序图的格式, 该图描述了字处理 (Word-Processing, WP) 软件产品中**确定概念范围**这一任务的软件项目进度安排。所有的项目任务(针对“确定概念范围”)都在左边栏中列出。水平条表示各个任务的工期, 当同一时段中存在多个水平条时, 就代表任务之间的并发性, 菱形表示里程碑。

输入了生成时序图所需的信息之后, 大多数软件项目进度安排工具都能生成项目表 (project table)——列出所有项目任务的表格, 项目表中列出了各个任务计划的开始与结束日期、实际开始日期与结束日期以及各种相关信息(图 34-4)。通过项目表与时序图, 项目管理者就可以跟踪项目的进展情况。

34.5.2 跟踪进度

如果制定正确, 项目进度表应该成为一个能够确定在项目进展过程中跟踪和控制任务及里程碑的线路图。项目跟踪可以通过以下方式实现:

关键点 通过时序图可以确定在特定时间点将进行什么任务。

引述 软件状态报告的基本规则可以归纳为一句话: 一点都不奇怪。
Capers Jones

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

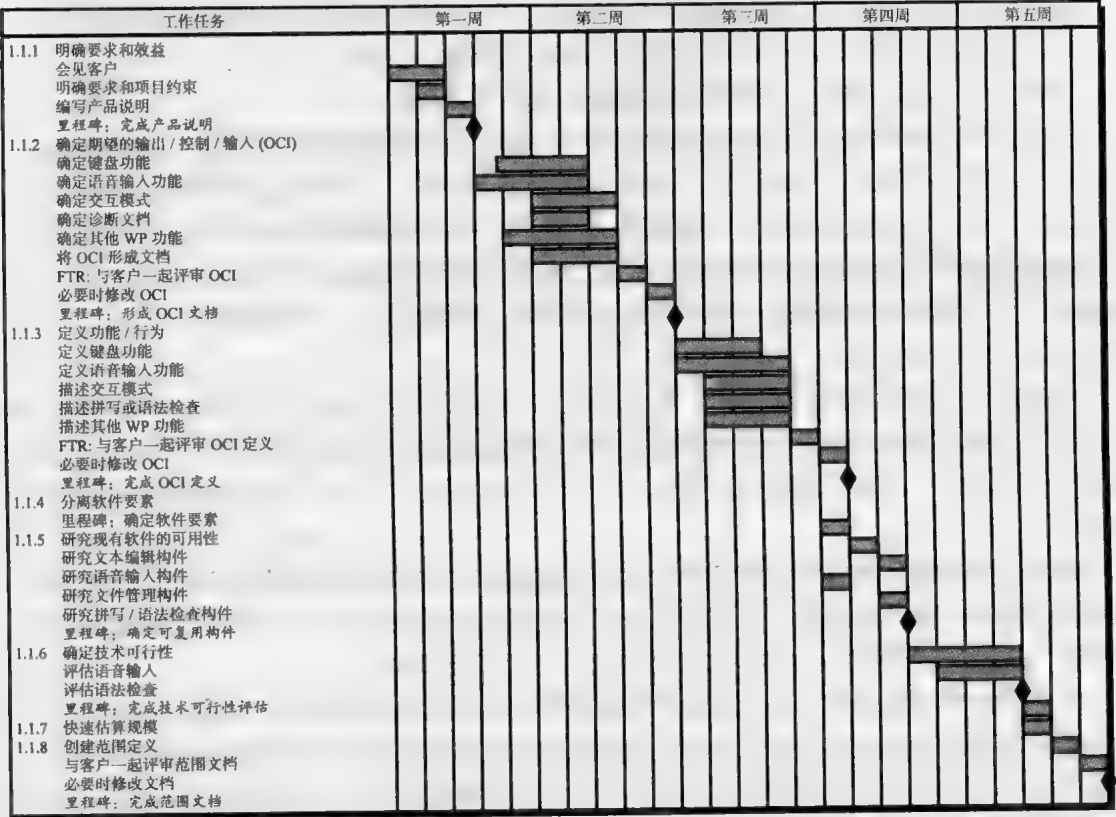


图 34-3 一个时序图的例子

工作任务	计划开始	实际开始	计划完成	实际完成	人员分配	工作量分配	备注
1.1.1 明确要求和效益							
会见客户	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
明确要求和项目约束	wk1, d2	wk1, d2	wk1, d3	wk1, d3	JPP	1 p-d	
编写产品说明	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
里程碑: 完成产品说明	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
1.1.2 确定期望的输出 / 控制 / 输入 (OCI)							
确定键盘功能	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
确定语音输入功能	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
确定交互模式	wk2, d1		wk2, d3		MLL	1 p-d	
确定诊断文档	wk2, d1		wk2, d2		BLS	1.5 p-d	
确定其他 WP 功能	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
将 OCI 形成文档	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: 与客户一起评审 OCI	wk2, d3		wk2, d3		all	3 p-d	
必要时修改 OCI	wk2, d4		wk2, d4		all	3 p-d	
里程碑: 形成 OCI 文档	wk2, d5		wk2, d5				
1.1.3 定义功能 / 行为							

图 34-4 一个项目表的例子

- 定期举行项目状态会议，由项目团队中的各成员分别报告进度和存在的问题。
- 评估在软件工程过程中所进行的所有评审的结果。
- 判断正式的项目里程碑（图 34-3 中的菱形）是否在预定日期内完成。
- 比较项目表（图 34-4）中列出的各项任务的实际开始日期与计划开始日期。

- 与开发人员进行非正式会谈，获取他们对项目进展及可能出现的问题的客观评估。
- 通过挣值分析（34-6 节）来定量地评估项目进展。

实际上，有经验的项目管理者会使用所有这些跟踪技术。

软件项目管理者通过施加控制来管理项目资源、处理问题和指导项目参与者。如果一切顺利（即项目在预算范围内按进度进行，评审结果表明的确取得了实际进展，达到了各个里程碑），则几乎不必施加控制。但是如果出现问题，项目管理者就必须施加控制，以便尽快解决问题。当诊断出问题之后，可能需要增加额外的资源来解决问题：雇用新员工或者重新安排项目进度。

建议 项目进展的最佳指标就是所定义的软件工作产品的实现和成功评审。

在面对交付期限的巨大压力时，有经验的项目管理者有时会使用一种称为时间盒（time-boxing）[Jal04] 的项目进度安排与控制技术。时间盒方法认为完整的产品可能难以在预定时间内交付，因此，应该选择增量软件开发范型（第 4 章），并为每个增量的交付制定各自的进度表。

接着，对与每个增量相关的任务实行时间盒技术。也就是按该增量的交付日期向后进行推算来调整各个任务的进度。将各个任务放入相应的“盒子”中，当一个任务触及其时间盒边界时（ $\pm 10\%$ 的范围内），则该项任务停止，下一任务开始。

对时间盒方法的最初反应通常是消极的：“如果工作尚未完成，我们该如何继续？”这个问题的答案在于完成工作的方式。当遇到时间盒的边界时，很可能已经完成了任务的 90%^①，余下 10% 的工作尽管重要，但是可以：（1）推迟到下一个增量中；或（2）在以后需要时再完成。项目朝着交付日期推进，而不是“卡”在某项任务上。

34.5.3 跟踪面向对象项目的进展

虽然迭代模型是最好的面向对象项目框架，但是，任务的并行性使得面向对象项目很难跟踪。困难在于项目管理者很难为面向对象项目建立有意义的里程碑，因为很多不同事物都是同时发生的。通常，有相应的准则来衡量下列主要的里程碑是否已经“完成”。

技术里程碑：面向对象分析完成

- 已经定义和评审了所有的类和类层次。
- 已经定义和评审了与每一个类相关的属性和操作。
- 已经建立和评审了各类之间的关系（第 10 章）。
- 已经建立和评审了行为模型（第 11 章）。
- 已经确定了可复用的类。

技术里程碑：面向对象设计完成

- 已经确定和评审了子系统集合。
- 各类已经分配给相应的子系统，并且已经通过评审。
- 已经建立和评审了任务分配。
- 已经明确责任和协作。
- 已经设计和评审了属性和操作。

① 爱冷嘲热讽的人也许会想起一句谚语：“完成系统的前 90% 需要 90% 的时间，完成剩下的 10% 也要用 90% 的时间。”

- 已经创建和评审了通信模型。

技术里程碑：面向对象程序设计完成

- 按照设计模型，每一个新类都已经编码实现。
- (从可复用库中) 提取的类已经实现。
- 已经构建了原型或增量。

技术里程碑：面向对象测试

- 已经评审了面向对象分析和设计模型的正确性和完整性。
- 已经建立和评审了类 - 职责 - 协作者网络 (第 10 章)。
- 已经设计了测试用例，并且已经对每个类进行了类级测试 (第 24 章)。
- 已经设计了测试用例，并且已经完成簇测试 (第 24 章)，已经完成类的集成。
- 已经完成系统级测试。

建议 调试和测试是相辅相成的，通常可以通过考察“公开的”错误 (缺陷) 类型和数量来判断调试的状态。

就像前面介绍的，建立面向对象过程模型是以迭代方式进行的，在交付不同的增量给用户时，上述的每一个里程碑都可以进行修订。

34.5.4 WebApp 和移动 App 项目的进度安排

WebApp 和移动 App 项目进度安排 (Web and MobileApp project scheduling) 就是将所估算的工作量分配到构建每个增量的计划时序 (项目工期) 内，这可以通过将工作量分配给那些特定的任务来完成。但要注意的是，整个进度是随时间而不断演化的。刚开始的时候，建立的是一张宏观进度表，该进度表标识了所有的 WebApp 或移动 App 增量以及每一个增量的计划部署日期。随着增量开发的进展，宏观进度表中与各个增量对应的条目会被细化成详细的进度表，这样就标识了 (完成一个活动所必须实现的) 特定开发任务，同时也进行了进度安排。

下面以 SafeHomeAssured.com 项目为例来更好地理解宏观进度安排。就像前面对 SafeHomeAssured.com 的讨论，基于 Web 的项目的构件可以定义为 7 个增量：

增量 1：企业基本情况与产品基本信息

增量 2：详细的产品信息与下载

增量 3：产品报价与产品订单处理

增量 4：空间布局与安全系统设计

增量 5：监控服务信息与监控服务排序

增量 6：监控设备的联机控制

增量 7：账户信息访问

软件团队与利益相关者共同磋商后，制定了针对这 7 个增量的初步的 (preliminary) 部署进度表。与该进度表对应的时序图如图 34-5 所示。

要注意的是，这里的部署日期 (时序图中的菱形) 只是大概的日期，在对增量做更详细的进度安排时可以进行修改。根据这个宏观进度表，管理者可以了解到什么时候能够得到内容和功能，以及整个项目什么时候可以完成。作为初步估算，软件团队将在 12 周的时间内部署所有增量。从图中还可以看出有些增量是并行开发的 (例如，增量 3、增量 4 和增量 7)，这里假定软件团队有足够的人员来完成并发的开发工作。

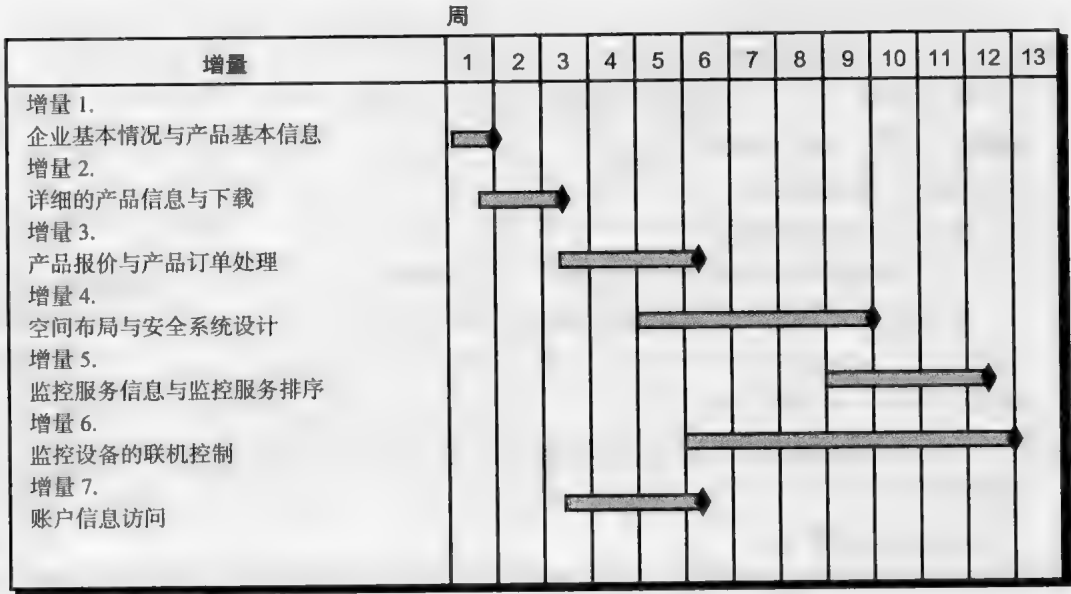


图 34-5 宏观项目进度表时序图

770

完成宏观进度表后，软件团队就可以开始为特定的增量安排工作任务。计划者可以采用适用于所有增量的通用过程框架来完成这项工作。首先，将框架中的通用任务提取出来组成任务列表（task list），然后，根据特定 WebApp 增量要提取的内容和功能来处理这些任务。

每一个框架行动（及其相关的任务）可以按下面 4 种方式之一进行处理：（1）应用任务；（2）删除任务，该任务对这个增量来说不重要；（3）增加新的（自定义）任务；（4）将任务细化（详细描述）为多个命名的子任务，并将这些子任务写入进度表。

例如，考虑一般的 WebApp 设计建模（design modeling）活动，完成这项活动需要注意到 WebApp 的一般设计任务，这些任务在第 17 章中讨论过。以 SafeHomeAssured.com 第 4 个增量的“界面设计”任务为例。就像前面介绍的，第 4 个增量要实现能够描述居住空间或营业空间的内容及功能，这些居住空间或营业空间将受到 SafeHome 安全系统的保护。根据图 34-5，第 4 个增量从第 5 周初开始，第 9 周末结束。

毫无疑问，必须完成界面设计任务。软件团队意识到界面设计是该增量能否成功的关键，因此决定对这个任务进行细化（详细描述）。下列子任务全部来源于第 4 个增量的界面设计任务：

- 绘制空间设计页面的页面布局草图。
- 与利益相关者一起评审布局。
- 空间布局导航机制设计。
- “绘图板”布局[⊖]设计。
- 给出墙壁绘图功能的程序细节。
- 给出墙壁的长度计算与显示功能的程序细节。
- 给出窗户绘图功能的程序细节。

⊖ 在这里，软件团队可以想象一下通过绘图功能真实地绘出空间的墙壁、窗户和门。墙壁线通常与网格点“对齐”，墙壁的大小会自动显示，窗户和门可以通过图形定位。空间创建好之后，最终用户还可以选择特定的传感器、摄像机等，并确定这些设备在空间内的位置。

- 给出门绘图功能的程序细节。
- 选择安全系统构件（传感器、摄像机、麦克风等）的机制设计。
- 给出安全系统构件绘图功能的程序细节。
- 必要时两人一组进行走查。

771

将这些子任务写入 WebApp 第 4 个增量的进度表，并分配到整个增量开发进度表中，这样就可以将这些子任务输入进度安排软件（如微软项目）中以在跟踪和控制时使用。

SafeHome 跟踪进度

[场景] Doug Miller 的办公室，SafeHome 软件项目开始之前。

[人物] Doug Miller (SafeHome 软件工程项目团队经理)、Vinod Raman、Jamie Lazar 以及产品软件工程团队的其他成员。

[对话]

Doug (看着 PPT): 第一个 SafeHome 增量的进度表看起来比较合理，但是，我们很难跟踪项目进展情况。

Vinod (看上去非常担心): 为什么？大部分工作产品的任务都是按天来安排进度的，并且我们保证并没有过度分配资源。

Doug: 一切都好。但是，我们怎样才能确定什么时候能完成第一个增量的分析模型呢？

Jamie: 任务是迭代的，所以很难。

Doug: 我知道，但是……好吧，例如，就拿“确定分析类”来说，你们认为它是一个里程碑。

Vinod: 是啊。

Doug: 是谁做的决定？

Jamie (很生气): 谁做的有什么关系？

Doug: Jamie，这样不太好。我们必须安排 TR (技术评审，第 20 章)，可是你还没做呢。例如，成功完成对分析模型的评审就是一个合理的里程碑。清楚了吗？

Jamie (皱着眉): 好吧，回到制图板。

Doug: 完成修正不能超过 1 个小时……现在其他人可以开始了。

34.6 挣值分析

在 34.5 节，我们讨论了一系列项目跟踪的定性方法，为项目管理者提供了项目进展情况的指标。但是，对所提供信息的评估在某种程度上是主观的。那么当软件团队按项目进度表实施工作任务时，是否存在某种定量的技术来评估项目进展情况呢？事实上，确实存在一种用于项目进展的定量分析技术，称为挣值分析 (Earned Value Analysis, EVA)。Humphrey[Hum95] 对挣值给出了如下讨论：

不管要完成何种类型的工作，挣值系统为每个（软件项目）任务提供了通用的值尺度，可以估算完成整个项目所需要的总小时数，并且可以根据各个任务所估算的小时数占总小时数的百分比来确定该任务的挣值。

更简单地说，挣值是对项目进展的测量。它使得计划者能够不依赖于感觉，而是采用定量的分析方法来评估一个项目的“完成百分比”。事实上，Fleming 和 Koppleman[Fle98] 就认为挣值分析“早在项目进展的前 15% 就提供了精确而可靠的性能数据”。按照以下步骤可以确定挣值。

关键点 挣值提供了定量的项目进展指标。

772

1. 为进度表中的每个工作任务确定其预计工作的预算成本 (Budgeted Cost of Work Scheduled, BCWS)。在估算过程中, 要计划每个软件工作任务的工作量 (以人时或人日为单位), 因此, $BCWS_i$ 是指工作任务 i 的计划工作量。为了确定在项目进度表中某特定时间点的项目进展状况, BCWS 的值是在项目进度表中该时间点应该完成的所有工作任务的 $BCWS_i$ 值之和。
2. 所有工作任务的 BCWS 值加起来, 可计算出完成工作的预算 (Budget at Completion, BAC), 因此, 对所有任务 k , 有

$$BAC = \sum(BCWS_k)$$

3. 接着, 计算已完成工作的预算成本 (Budgeted Cost of Work Performed, BCWP)。BCWP 的值是在项目进度表中该时间点已经实际完成的所有工作任务的 BCWS 值之和。

Wilkens[Wil99] 指出: “BCWS 和 BCWP 的不同点是, 前者表示计划将完成的工作的预算, 后者表示已实际完成的工作的预算。” 给定 BCWS、BAC 和 BCWP 的值, 就可以得出相关的项目进展指标:

$$\text{进度表执行指标 SPI} = \frac{BCWP}{BCWS}$$

$$\text{进度表偏差 SV} = BCWP - BCWS$$

其中, SPI 是效率指标, 指出项目使用预定资源的效率, SPI 值越接近 1.0 说明项目的执行效率越高。SV 只表示与计划进度的偏差。

$$\text{预定完成百分比} = \frac{BCWS}{BAC}$$

表示在时间点 t 应该完成工作的百分比值。

$$\text{完成百分比} = \frac{BCWS}{BAC}$$

773 表示在特定时间点 t 实际完成工作的百分比值。

也可以计算出已完成工作的实际成本 (Actual Cost of Work Performed, ACWP)。ACWP 的值是在项目进度表中某时间点已经完成的工作任务的实际工作量之和。然后, 再计算:

$$\text{成本执行指标 CPI} = \frac{BCWP}{ACWP}$$

$$\text{成本偏差 CV} = BCWP - ACWP$$

CPI 值越接近 1.0 说明项目与预算越接近。CV 表示在项目特定阶段的成本节省 (相对于计划成本) 或短缺。

就像超视距雷达一样, 挣值分析在可能出现问题之前就指出了进度安排的难点, 这使得软件项目管理者能够在项目危机出现前采取有效措施。

34.7 小结

计划活动是软件项目管理的重要组成部分, 而进度安排是计划活动的首要任务。进度安排与估算方法及风险分析相结合, 可以为项目管理者画出一张路线图。

进度安排始于过程分解。根据项目特性, 为将要完成的工作选择适当的任务集。任务网

提问 如何计算挣值以评估项目进展。

网络资源 有关挣值分析的信息源见 <http://www.acq.osd.mil/evm/>。

络描述了各项工程任务、每一项任务与其他任务之间的依赖关系以及计划工期。任务网络可以用来确定项目的关键路径、时序图以及各种项目信息。以进度表为指导,项目管理者可以跟踪和控制软件工程过程中的每一个步骤。

习题与思考题

- 34.1 “不合理的”项目最后期限是软件行业中存在的现实情况。当你遇到这种情况时应该如何处理?
- 34.2 宏观进度表和详细进度表的区别是什么?是否有可能只依据所制定的宏观进度表来管理一个项目?为什么?
- 34.3 是否存在这种情况:一个软件项目里程碑没有与某个评审相关联?如果有,请至少给出一个例子。
- 34.4 当多个人员参与软件项目时,就有可能产生“交流开销”。与其他人员进行交流要花费时间,这样就会降低个人生产率(LOC/月),最终导致整个团队生产率下降。(举几个例子)量化说明非常精通软件工程实践和运用技术评审的软件工程师是如何提高团队生产率的(与个人生产率的总和进行比较)。提示:假设评审减少了返工,而返工可能占一个人 20%~40% 的时间。
- 34.5 尽管为延迟的软件项目增加人手可能会进一步拖延工期,但是否在某些情况下并非如此呢?请说明。
- 34.6 人员和时间的关系是高度非线性的。使用 Putnam 的软件方程(34.2.2 节)编制一个表,以反映软件项目中人员数量与项目工期之间的关系——该项目需要 50000 LOC 和 15 人年的工作量(生产率参数为 5000, $B=0.37$)。假定该软件必须在 24 ± 12 个月的时间期限内交付。
- 34.7 假定你要为一所大学开发一个联机课程登记系统(Online Course Registration System, OLCRS)。首先从客户的角度(如果你是一名学生就很容易了!)指出一个好系统应该具有的特性。(或者你的老师会为你提供一些初步的系统需求。)按照第 33 章所介绍的估算方法,估算 OLCRS 系统的开发工作量和工期。建议你按如下方式进行:
- 确定 OLCRS 项目中的并行工作活动。
 - 将工作量分布到整个项目中。
 - 建立项目里程碑。
- 34.8 为 OLCRS 项目选择适当的任务集。
- 34.9 为 34.7 题中的 OLCRS 或者你感兴趣的其他软件项目定义任务网络。确信你已给出所有的任务和里程碑,并为每一项任务分配了所估算的工作量和工期。如果可能的话,使用自动进度安排工具来完成这一工作。
- 34.10 如果有自动进度安排工具,请为习题 34.9 中定义的任务网络确定关键路径。
- 34.11 使用进度安排工具(如果有条件)或者纸笔(如果需要)制定 OLCRS 项目的时序图。
- 34.12 假设你是一个软件项目管理者,受命为一个小型软件项目进行挣值统计。这个项目共计划了 56 个工作任务,估计需要 582 人日才能完成。目前有 12 个工作任务已经完成,但是,按照项目进度,现在应该完成 15 个任务,下面给出相关进度安排数据(单位:人日),请你做出挣值分析。

任务	计划工作量	实际工作量	任务	计划工作量	实际工作量
1	12.0	12.5	9	12.0	10.0
2	15.0	11.0	10	6.0	6.5
3	13.0	17.0	11	5.0	4.0
4	8.0	9.5	12	14.0	14.5
5	9.5	9.0	13	16.0	—
6	18.0	19.0	14	6.0	—
7	10.0	10.0	15	8.0	—
8	4.0	4.5			

计算该项目的进度表执行指标 SPI、进度表偏差 SV、预定完成百分比、完成百分比、成本执行指标 CPI、成本偏差 CV。

扩展阅读与信息资源

事实上，每一本关于软件项目管理的书都会包含进度安排的内容。项目管理研究所（《PMBOK Guide》，5th ed., PMI, 2013）、Wysoki（《Effective Project Management: Traditional, Agile, Extreme》，6th ed., Wiley, 2011）、Lewis（《Project Planning Scheduling and Control》，5th ed., McGraw-Hill, 2010）、Kerzner（《Project Management: A Systems Approach to Planning, Scheduling, and Controlling》，10th ed., Wiley, 2009）、Chemuturi 和 Cagley（《Mastering Software Project Management: Best Practices, Tools, and Techniques》，J. Ross Publishing, 2010）、Hughes 和 Cotterel（《Software Project Management》，5th ed., McGraw-Hill, 2009）、Luckey 和 Phillips（《Software Project Management for Dummies》，For Dummies, 2006）、Lewin（《Better Software Project Management》，Wiley, 2001）以及 Bennatan（《On Time, Within Budget: Software Project Management Practices and Techniques》，3rd ed., Wiley, 2000）都包含了对这一主题的有价值的讨论。尽管 Harris（《Planning and Scheduling Using Microsoft Office Project 2010》，Eastwood Harris Pty Ltd., 2010）只是一个应用特例，但是它深入探讨了如何有效地利用进度安排工具对软件项目进行跟踪和控制。

Fleming 和 Koppelman（《Earned Value Project Management》，3rd edition, Project Management Institute Publications, 2010）、Budd（《A Practical Guide to Earned Value Project Management》，2nd ed., Management Concepts, 2009）以及 Webb 和 Wake（《Using Earned Value: A Project Manager's Guide》，Ashgate Publishing, 2003）较详细地讨论了挣值技术在项目计划、跟踪和控制方面的应用。

网上有大量的与软件项目进度安排相关的信息资源。最新的参考文献参见 SEPA 网站 www.mhhe.com/pressman 下的“software engineering resources”。

风险管理

要点浏览

概念: 很多问题都会困扰软件项目, 风险分析和风险管理就是辅助软件团队理解和管理不确定事物的一系列步骤。风险是潜在的——它可能发生也可能不发生。但是, 不管发生还是不发生, 都应该去识别它, 评估它发生的概率, 估算它的影响, 并制定它实际发生时的应急计划。

人员: 软件过程中涉及的每一个人——管理者、软件工程师和利益相关者——都要参与风险分析和风险管理。

重要性: 想想童子军的格言: “时刻准备着。” 软件项目是困难重重的任务, 很多事情都可能出错, 而且坦率地说, 很多事情经常出错。为此, 时刻准备着——理解风险、采取主动的措施去回避或管理风险——是一个优秀的软件项目管理者应具备的基本条件。

步骤: 第一步称为“风险识别”, 即辨别出什么情况下可能会出问题。第二步, 分析每个风险, 确定其可能发生的概率以及发生时将带来的危害。了解这些信息之后, 就可以按照可能发生的概率和危害程度对风险进行排序。第三步, 制定一个计划来管理那些发生概率高和危害程度大的风险。

工作产品: 风险缓解、监测和管理 (Risk Mitigation, Monitoring and Management, RMMM) 计划或一组风险信息表单。

质量保证措施: 所要分析和管理的风险, 应该经过对人员、产品、过程和项目的彻底研究后再确定。RMMM 计划应该随着项目的进展而修订, 以保证所考虑的风险是近期可能发生的。风险管理的应急计划应该是符合实际的。

Robert Charette[Cha89] 在他关于风险分析与管理的书中给出了风险概念的定义:

首先, 风险涉及的是未来将要发生的事情。今天和昨天的事情已不再关心, 如同我们已经在收获由我们过去的行为所播下的种子。问题是: 我们是否能够通过改变今天的行为, 而为一个不同的、充满希望的、更美好的明天创造机会。其次, 风险涉及改变。如思想、观念、行为、地点的改变……第三, 风险涉及选择, 而选择本身就具有不确定性。因此, 就像死亡和税收一样, 风险是生活中最不确定的因素之一。

对于软件工程领域中的风险, Charette 的三条概念定义是显而易见的。未来是项目管理者所关心的——什么样的风险会导致软件项目彻底失败? 改变也是项目管理者所关心的——客户需求、开发技术、目标环境以及所有其他与项目相关因素的改变将会对进度安排和总体成功产生什么影响?

关键概念

评估

识别

预测

细化

风险分类

风险显露度

风险条目检查表

风险表

RMMM

安全和灾难

策略

主动策略

被动策略

最后，项目管理者必须抓住选择机会——应该采用什么方法及工具？需要多少人员参与？对质量的要求要达到什么程度才是“足够的”？

Peter Drucker [Dru75] 曾经说过：“当没有办法消除风险，甚至连试图降低该风险也存在疑问时，这个风险就是真正的风险了。”在弄清楚软件项目中的“真正风险”之前，识别出所有对管理者及开发者而言显而易见的风险是很重要的。

35.1 被动风险策略和主动风险策略

被动风险策略 (reactive risk strategy) 被戏称为“印第安纳·琼斯学派的风险管理” [Tho92]。印第安纳·琼斯在以其名字命名的电影中，每当面临无法克服的困难时，总是一成不变地说：“不要担心，我会想出办法来的！”印第安纳·琼斯从不担心任何问题，直到风险发生，再做出英雄式的反应。

引述 如果你不主动进攻风险，风险将会主动进攻你。

Tom Gilb

遗憾的是，一般的软件项目管理者并不是印第安纳·琼斯，软件项目团队的成员也不是他可信赖的伙伴。因此，大多数软件项目团队还是仅仅依赖于被动的风险策略。被动策略最多不过是针对可能发生的风险来监测项目，直到风险发生时，才会拨出资源来处理它们。大多数情况下，软件项目团队对风险不闻不问，直到出现了问题。这时，项目团队才赶紧采取行动，试图迅速地纠正错误，这通常叫作救火模式 (fire-fighting mode)。当这样的努力失败后，“危机管理” [Cha92] 接管一切，这时项目已经处于真正的危机中了。

对于风险管理，更好的是主动风险策略。主动 (proactive) 风险策略早在技术工作开始之前就已经启动了。识别出潜在的风险，评估它们发生的概率及产生的影响，并按其重要性进行排序。然后，软件项目团队就可以制定一个计划来管理风险。计划的主要目标是回避风险，但不是所有的风险都能够回避，所以，项目团队必须制定一个应急计划，使其在必要时能够以可控和有效的方式做出反应。在本章的后面将讨论风险管理的主动策略。

35.2 软件风险

虽然对于软件风险的严格定义还存在很多争议，但一般认为软件风险包含两个特性 [Hig95]：不确定性 (uncertainty) 是指风险可能发生也可能不发生，即没有 100% 会发生的风险^①；损失 (loss) 是指如果风险发生，就会产生恶性后果或损失。进行风险分析时，重要的是量化每个风险的不确定程度和损失程度。为了实现这一点，必须考虑不同类型的风险。

项目风险 (project risk) 威胁到项目计划。也就是说，如果项目风险发生，就有可能拖延项目的进度和增加项目的成本。项目风险是指预算、进度、人员 (聘用职员及组织)、资源、利益相关者、需求等方面的潜在问题以及它们对软件项目的影响。在第 33 章中，项目复杂度、规模及结构不确定性也属于项目 (和估算) 风险因素。

提问 在构建软件时可能会遇到什么类型的风险？

技术风险 (technical risk) 威胁到要开发软件的质量及交付时间。如果技术风险发生，开发工作就可能变得很困难或根本不可能。技术风险是指设计、实现、接口、验证和维护等方面的潜在问题。此外，规格说明的歧义性、技术的不确定性、技术陈旧以及“前沿”技术

① 100% 发生的风险是强加在软件项目上的约束。

也是技术风险因素。技术风险的发生是因为问题比我们所设想的更加难以解决。

商业风险 (business risk) 威胁到要开发软件的生存能力, 且常常会危害到项目或产品。五个主要的商业风险是: (1) 开发了一个没有人真正需要的优良产品或系统 (市场风险); (2) 开发的产品不再符合公司的整体商业策略 (策略风险); (3) 开发了一个销售部门不知道如何去销售的产品 (销售风险); (4) 由于重点的转移或人员的变动而失去了高级管理层的支持 (管理风险); (5) 没有得到预算或人员的保证 (预算风险)。

应该注意的是, 单一的风险分类并不总是行得通, 有些风险根本无法事先预测。

另一种常用的风险分类方式是由 Charette [Cha89] 提出的。已知风险 (known risk) 是通过仔细评估项目计划、开发项目的商业及技术环境以及其他可靠的信息来源 (如不现实的交付时间, 没有文档化需求或软件范围、恶劣的开发环境) 之后可以发现的那些风险。可预测风险 (predictable risk) 能够从过去项目的经验中推断出来 (如人员变动、与客户之间欠缺沟通、由于正在进行维护而使开发人员精力分散)。不可预测风险 (unpredictable risk) 就像纸牌中的大王, 它们可能会真的出现, 但很难事先识别。

引述 不经历实际风险的项目是不可能成功的。这种项目几乎是无益的, 否则早就有人开发了。

Tom DeMarco,

Tim Lister

779

信息栏 风险管理的 7 个原则

美国卡内基·梅隆大学软件工程研究所 (Software Engineering Institute, SEI, www.sei.cmu.edu) 定义了“实施有效风险管理框架”的 7 条原则:

保持全面观点——在软件所处的系统中考虑软件风险以及该软件所要解决的业务问题。

采用长远观点——考虑将来可能发生的风险 (如软件的变更), 并制定应急计划使将来发生的事件成为可管理的。

鼓励广泛交流——如果有人提出一个潜在的风险, 要重视它; 如果以非正式的方式提出一个风险, 要考虑它。任何时候都要

鼓励利益相关者和用户提出风险。

结合——考虑风险时必须与软件过程相结合。

强调持续的过程——在整个软件过程中, 团队必须保持警惕。随着信息量的增加, 要修改已识别的风险; 随着知识的积累, 要加入新的风险。

开发共享的产品——如果所有利益相关者共享相同版本的软件产品, 将更容易进行风险识别和评估。

鼓励协同工作——在风险管理活动中, 要汇聚所有利益相关者的智慧、技能和知识。

35.3 风险识别

风险识别试图系统化地指出对项目计划 (估算、进度、资源分配等) 的威胁。识别出已知风险和可预测风险后, 项目管理者首先要做的是在可能时回避这些风险, 在必要时控制这些风险。

35.2 节中提出的每一类风险又可以分为两种不同的类型: 一般风险和产品特定的风险。一般风险 (generic risk) 对每一个软件项目而言都是潜在的威胁。而产品特定的风险 (product-specific risk) 则只有那些对当前项目特定的技术、人员及环境非常了解的人才能识

别出来。为了识别产品特定的风险，必须检查项目计划及软件范围说明，然后回答这个问题：“本产品中有什么特殊的特性可能会威胁到我们的项目计划？”

识别风险的一种方法是建立风险条目检查表。该检查表可用于风险识别，并且主要用来识别下列几种类型中的一些已知风险和可预测风险：

- 产品规模 (product size) ——与要开发或要修改的软件的总体规模相关的风险。
- 商业影响 (business impact) ——与管理者或市场所施加的约束相关的风险。
- 项目相关人员特性 (stakeholder characteristic) ——与项目相关人员的素质以及开发者和项目相关人员定期沟通的能力相关的风险。
- 过程定义 (process definition) ——与软件过程定义的程度以及该过程被开发组织遵守的程度相关的风险。
- 开发环境 (development environment) ——与用来开发产品的工具的可得性及质量相关的风险。
- 开发技术 (technology to be built) ——与待开发软件的复杂性及系统所包含技术的“新奇性”相关的风险。
- 人员才干及经验 (staff size and experience) ——与软件工程师的总体技术水平及项目经验相关的风险。

风险条目检查表可以采用不同的方式来组织。与上述每个主题相关的问题可以针对每一个软件项目来回答。有了这些问题的答案，项目管理者就可以估计风险产生的影响。也可以采用另一种不同的风险条目检查表格式，即仅仅列出与每一种类型有关的特性。最终，给出一组“风险因素和驱动因子”[AFC88]以及它们发生的概率。有关性能、支持、成本及进度的驱动因子将在后面进行讨论。

网上有很多针对软件项目风险的检查表（例如，[Baa07]、[Nas07]、[Wor04]），项目管理者可以利用这些检查表来提高识别软件项目一般风险的洞察力。除了使用清单，风险模式[Mil04]也可作为系统的风险识别方法。

35.3.1 评估整体项目风险

下面的提问来源于对世界各地有经验的软件项目管理人员的调查而得到的风险资料[Kei98]，根据各个问题对项目成功的相对重要性将问题进行了排序。

1. 高层的软件管理者和客户管理者已经正式承诺支持该项目了吗？
2. 最终用户对项目和待开发的系统或产品热心支持吗？
3. 软件工程团队及其客户充分理解需求了吗？
4. 客户已经完全地参与到需求定义中了吗？
5. 最终用户的期望现实吗？
6. 项目范围稳定吗？
7. 软件工程团队的技能搭配合理吗？
8. 项目需求稳定吗？

建议 虽然考虑一般风险很重要，但是，通常产品特定的风险会带来更多的问题。一定要花时间尽可能多地识别出产品特定的风险。

提问 我们正在进行的软件项目面临严重的风险吗？

网络资源 风险雷达 (risk radar) 是一种数据库，也是一种工具，它可以帮助项目管理人员识别、排序和交流项目风险，见 www.spmn.com。

9. 项目团队对将实现的技术有经验吗?
10. 项目团队的人员数量满足项目需要吗?
11. 所有的客户或用户对项目的重要性和待开发的系统或产品的需求有共识吗?
- 如果对这些问题的任何一个回答是否定的, 则应务必启动缓解、监测和管理风险的步骤。项目的风险程度与对这些问题否定回答的数量成正比。

35.3.2 风险因素和驱动因子

美国空军有一本小册子 [AFC88], 其中包含了如何很好地识别和消除软件风险的指南。他们所用的方法是要求项目管理者识别影响软件风险因素的风险驱动因子, 风险因素包括: 性能、成本、支持和进度。在这里, 风险因素是以如下的方式定义的:

引述 风险管理是针对成年人的项目管理。
Tim Lister

- 性能风险 (performance risk) ——产品能够满足需求且符合其使用目的的不确定程度。
 - 成本风险 (cost risk) ——能够维持项目预算的不确定程度。
 - 支持风险 (support risk) ——开发出的软件易于纠错、修改及升级的不确定程度。
 - 进度风险 (schedule risk) ——能够维持项目进度且按时交付产品的不确定程度。
- 每一个风险驱动因子对风险因素的影响均可分为四个影响类别: 可忽略的、轻微的、严重的或灾难的。图 35-1[Boe89] 指出了由于未识别出的软件失误而产生的潜在影响 (标号为 1 的行), 或没有达到预期的结果所产生的潜在影响 (标号为 2 的行)。影响类别的选择是以最符合表中描述的特征为基础的。

风险因素 影响类别		性能	支持	成本	进度
灾难的	1	无法满足需求而导致任务失败		失误将导致进度延迟和成本增加, 预计超支 \$500K	
	2	严重退化使得根本无法达到要求的技术性能	无法做出响应或无法支持的软件	资金严重短缺, 很可能超出预算	无法在交付日期内完成
严重的	1	无法满足需求而导致系统性能下降, 使得任务能否成功受到质疑		失误将导致系统运行的延迟并使成本增加, 预计超支 \$100K 到 \$500K	
	2	技术性能有些降低	在软件修改中有少量的延迟	资金不足, 可能会超支	交付日期可能延迟
轻微的	1	无法满足需求而导致次要任务的降级		成本、影响和可以补救的进度延迟上的小问题, 预计超支 \$1K 或 \$100K	
	2	技术性能有些降低	较敏感的软件支持	有充足的资金来源	现实的、可完成的进度计划
可忽略的	1	无法满足需求而导致使用不方便或不易操作		失误对进度和成本的影响很小, 预计超支少于 \$1K	
	2	技术性能没有降低	易于进行软件支持	可以低于预算	交付日期将会提前

注: 1. 未识别出的软件失误或缺陷所产生的潜在影响。
2. 如果没有到达预期的结果所产生的潜在影响。

图 35-1 影响评估 [Boe89]

35.4 风险预测

风险预测 (risk projection) 又称风险估计 (risk estimation), 试图从两个方面评估每一个风险: (1) 风险发生的可能性或概率; (2) 如果风险发生, 风险相关问题产生的后果。项目计划人员、其他管理人员及技术人员都要进行以下 4 步风险预测活动:

782

- 1. 建立一个尺度, 以反映风险发生的可能性。
- 2. 描述风险产生的后果。
- 3. 估算风险对项目及产品的影响。
- 4. 标明风险预测的整体精确度, 以免产生误解。

按此步骤进行风险预测, 目的是使我们可以按照优先级来考虑风险。任何软件团队都不可能以同样的严格程度来为每个可能的风险分配资源, 通过将风险按优先级排序, 软件团队可以把资源分配给那些具有最大影响的风险。

35.4.1 建立风险表

风险表给项目管理者提供了一种简单的风险预测方法[⊖]。风险表样本如图 35-2 所示。

风险	类型	发生概率	影响值	RMMM
规模估算可能很不正确	PS	60%	2	
用户数量大大超出计划	PS	30%	3	
复用程度低于计划	PS	70%	2	
最终用户抵制该系统	BU	40%	3	
交付期限太紧	BU	50%	2	
资金将会流失	CU	40%	1	
用户将改变需求	PS	80%	2	
技术达不到预期的效果	TE	30%	1	
缺少对工具的培训	DE	80%	3	
人员缺乏经验	ST	30%	2	
人员变动比较频繁	ST	60%	2	
.....				

影响值:
1—灾难的 2—严重的 3—轻微的 4—可忽略的

图 35-2 排序前的风险表样本

项目管理者首先要在表中的第一列列出所有风险 (不管多么细微)。这可以利用 35.3 节所述的风险条目检查表来完成。在第二列中给出每一个风险的类型 (例如, PS 指产品规模风险, BU 指商业影响风险)。在第三列中填入各个风险发生的概率。各个风险的概率值可以首先由团队成员各自估算, 然后按循环投票的方式进行, 直到大家对风险概率的评估值趋于接近为止。

下一步是评估每个风险所产生的影响。使用图 35-1 所示的特性评估每

建议 尽力思考你将要开发的软件, 并问自己: “估计会出什么问题?” 建立你自己的列表, 并要求团队的其他成员也这么做。

⊖ 风险表可以采用电子表格模式来实现, 使表中的条目易于操作和排序。

个风险因素，并确定其影响类别。将4个风险因素——性能、支持、成本及进度——的影响类别求平均^①可得到一个整体的影响值。

完成了风险表的前4列内容之后，就可以按照概率和影响值来进行排序。高概率、高影响的风险放在表的上方，而低概率风险则移到表的下方。这样就完成了第一次风险排序。

项目管理者研究排序后的表，然后定义一条中截线。该中截线(cutoff-line，表中某处之上的一条水平线)表示：只有那些在中截线之上的风险才会得到进一步的关注，而在中截线之下的风险则需要重新评估以进行第二次排序。参照图35-3，从管理关注的角度来看，风险的影响和发生的概率是截然不同的。一个具有高影响但发生概率很低的风险因素不应该耗费太多的管理时间，而高影响且发生概率为中到高的风险，以及低影响且高概率的风险，则应该首先列入随后的风险分析步骤中。

所有在中截线之上的风险都必须进行管理。标有RMMM的列中包含了一个指示器，指向为所有中截线之上的风险所建立的风险缓解、监测及管理计划(Risk Mitigation, Monitoring and Management Plan, RMMM计划)或一组风险信息表单。RMMM计划和风险信息表单将在35.7节讨论。

风险概率可以通过先做个别估计而后求出一个有代表性的值来确定。虽然这个方法可行的，不过还有很多其他更复杂的确定风险概率的技术(如[McC09])。

35.4.2 评估风险影响

如果风险真的发生了，那么有三个因素可能会影响风险所产生的后果，即风险的本质、范围和时间。风险的本质是指当风险发生时可能带来的问题。例如，一个定义很差的与客户硬件的外部接口(技术风险)会妨碍早期的设计和测试，也有可能导项目后期阶段的系统集成问题。风险的范围包括风险的严重性(即风险有多严重)及风险的整体分布情况(项目有多少部分受到影响或有多少客户受到损害)。最后，风险的时间是指何时能够感受到风险的影响及风险的影响会持续多长时间。在大多数情况下，项目管理者希望“坏消息”越早出现越好，但在某些情况下则是越迟越好。

让我们再回到美国空军提出的风险分析方法[AFC88]上来。下面的步骤可以用来确定风险的整体影响：(1)确定每个风险因素发生的平均概率。(2)使用图35-1中列出的标准来确定每个因素的影响。(3)按照前面几节给出的方法填写风险表，并分析其结果。

关键点 在风险表中应该按照概率和影响值对风险进行排序。

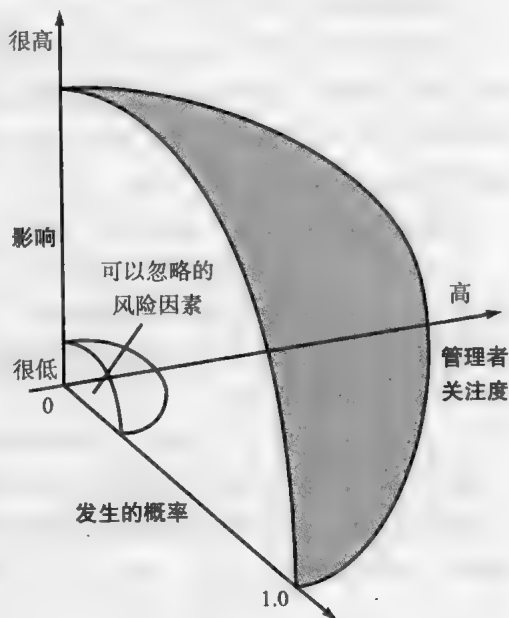


图 35-3 风险与管理

引述 如今，谁也不会奢望能够很好地了解任务以使其不会使人感到惊讶，而惊讶就意味着风险。

Stephen Grey

① 如果某个风险因素对项目来说比较重要，则可以使用加权平均法。

整体的风险显露度 (Risk Exposure, RE) 可由下面的关系确定 [HAL98]:

$$RE = P \times C$$

其中 P 是风险发生的概率, C 是风险发生时带来的项目成本。

例如, 假设软件团队按如下方式定义了项目风险。

风险识别: 事实上, 计划可复用的软件构件中只有 70% 将集成到应用中, 其他功能必须定制开发。

风险概率: 80% (大约)。

风险影响: 计划了 60 个可复用的软件构件, 如果只能利用 70%, 则 18 个构件必须从头开发 (除了已经计划开发的定制软件外)。平均每个构件的程序行数是 100 LOC, 本地数据表明每个 LOC 的软件工程成本是 \$14.00, 开发构件的总成本 (影响) 将是 $18 \times 100 \times 14 = \$25\,200$ 。

风险显露度: $RE = 0.80 \times \$25\,200 \approx \$20\,200$

风险的成本估算完成之后, 就可以为风险表中的每个风险计算其风险显露度。所有风险 (风险表中截线之上) 的总体风险显露度不仅为调整项目最终的成本估算提供了依据, 还可预测在项目进展过程中不同阶段所需人员资源的增长情况。

35.4.1 节和 35.4.2 节所述的风险预测和分析方法可以在软件项目进展过程中反复运用^①。项目团队应该定期复查风险表, 重新评估每一个风险, 以确定新环境是否引起其概率和影响发生改变。这样可能需要在风险表中添加一些新的风险, 删除一些不再有风险, 或改变一些风险的相对位置。

提问 如何评估风险产生的后果?

建议 将所有风险的 RE 与项目的成本估算进行比较。如果 RE 大于项目成本的 50%, 则必须重新评估项目的生存能力。

SafeHome 风险分析

[场景] Doug Miller 的办公室, SafeHome 软件项目开始之前。

[人物] Doug Miller (SafeHome 软件团队经理)、Vinod Raman、Jamie Lazar 以及产品软件工程团队的其他成员。

[对话]

Doug: 很高兴今天和大家一起讨论 SafeHome 项目的风险问题。

Jamie: 是讨论什么情况下可能会出问题吗?

Doug: 是的。这儿有几种可能会出问题的类型。(他给每个人展示了 35.3 节中给出的类型。)

Vinod: 嗯……你只是要求我们找出风险,

还是……

Doug: 不, 我想让每一个人建立一个风险列表, 立即动手……

(10 分钟过去了, 每个人都在写着。)

Doug: 好了, 停下来。

Jamie: 可是我还没有完成!

Doug: 没关系, 我们还要对列表进行复查。现在, 给列表中的每一个风险指定其发生概率的百分比值, 然后按 1 (较小的) 到 5 (灾难的) 的取值范围确定其对项目的影响。

Vinod: 就是说如果我认为风险的发生跟掷硬币差不多, 就给 50% 的概率, 如果我认为风险的影响是中等的, 就给影响

① 如果你有兴趣, 可阅读 [Ben10] 中给出的风险成本数学处理的相关内容。

值为3, 对吗?

Doug: 非常正确。

(5分钟过去了, 每个人都在写着。)

Doug: 好了, 停下来。现在, 我们在白板上建立一组列表, 我来写, 轮流从你们各自的列表中取出一项。

(15分钟过去了, 列表完成。)

Jamie (指着白板并笑着说): Vinod, 那个风险 (指向白板中的一项) 很可笑, 大家意外选中它的可能性很大, 应该删除。

Doug: 不, 先留着吧。不管有多么不可思议, 我们应考虑所有风险。一会儿我们还要精简这个列表。

Jamie: 已经有40多个风险了, 我们究竟怎样才能管理它们呢?

Doug: 管理不了。所以将这些风险排序之后, 我们还要定义中截线。明天我们继续开会讨论中截线。现在, 回去继续工作, 工作之余考虑是否还有遗漏的风险。

35.5 风险细化

在项目计划的早期, 风险很可能只是一个大概的描述。随着时间的推移, 我们对项目和风险的了解加深, 可以将风险细化为一组更详细的风险, 在某种程度上, 这些风险更易于缓解、监测和管理。

实现方法之一是按条件-转化-结果 (Condition-Transition-Consequence, CTC) 格式 [GLU94] 来表示风险, 即采用如下方式来描述风险:

提问 用什么好方式来描述风险?

787

给定 <条件>, 则 (可能) 导致 <结论>

使用 CTC 格式, 在 35.4.2 节中提到的可复用软件构件的风险可描述为:

给定条件: 所有可复用软件构件必须符合特定设计标准, 但是某些并不符合。则有结论: (可能) 仅 70% 的计划可复用构件将集成到最终的系统中, 需定制开发剩余 30% 的构件。

可按如下方式对这个一般条件进行细化:

子条件 1. 某些可复用构件是由第三方开发的, 没有其内部设计标准的相关资料。

子条件 2. 构件接口的设计标准尚未确定, 有可能和某些现有的软件可复用构件不一致。

子条件 3. 某些可复用构件是采用不支持目标环境的语言开发的。

这些子条件的结论是相同的 (即必须定制开发 30% 的软件构件), 但细化过程可以帮助我们排除重大风险, 使我们更易于分析风险和采取措施。

35.6 风险缓解、监测和管理

这里讨论的所有风险分析活动只有一个目的——辅助项目团队制定处理风险的策略。一个有效的策略必须考虑三个问题: 风险回避、风险监测、风险管理及应急计划。

如果软件团队采取主动的方法, 最好的策略就是风险回避。这可以通过建立一个风险缓解 (risk mitigation) 计划来实现。例如, 假设频繁的人员变动被标注为项目风险 r_1 。基于以往的历史和管理经验, 可以估计频繁人员变动的概率 l_1 为 0.70 (70%, 相当高), 并预测影响 x_1 为严重的。也就是说, 频繁的人员变动将对项目成本及进度有严重的影响。

引述 我采取如此多的预防措施, 是因为我不想冒任何风险。

Napoleon

为了缓解这个风险，项目管理者必须制定一个策略来减少人员变动。可能采取的步骤包括：

- 与现有人员一起探讨人员变动的起因（如恶劣的工作条件、报酬低、竞争激烈的劳动力市场）。
提问 如何缓解风险？
- 在项目开始之前采取行动，设法缓解那些我们能够控制的起因。
- 项目启动之后，假设会发生人员变动，当人员离开时，找到能够保证工作连续性的方法。
- 组织项目团队，使得每一个开发活动的信息能被广泛传播和交流。
- 制定工作产品标准，并建立相应机制以确保能够及时创建所有的模型和文档。
- 同等对待所有工作的评审（使得不止一个人能够“跟上进度”）。
- 给每一个关键的技术人员都指定一个后备人员。

随着项目的进展，风险监测（risk-monitoring）活动开始了，项目管理者应该监测那些可以表明风险是否正在变高或变低的因素。在人员变动频繁的例子中，应该监测：团队成员对项目压力的普遍态度、团队的凝聚力、团队成员彼此之间的关系、与报酬和利益相关的潜在问题、在公司内及公司外工作的可能性。

除了监测上述因素之外，项目管理者还应该监测风险缓解步骤的效力。例如，前面叙述的风险缓解步骤中要求制定工作产品标准，并建立相应机制以确保能够适时开发出工作产品。万一有关键成员离开此项目，应该有一个保证工作连续性的机制。项目管理者应该仔细监测这些工作产品，以保证每一个工作产品的正确性，在项目进行中有新员工加入时，能为他们提供必要的信息。

风险管理及应急计划（risk management and contingency planning）是以缓解工作已经失败而且风险已经发生为先决条件的。继续前面的例子，假定项目正在进行之中，有一些人宣布将要离开。如果已经按照缓解策略行事，则有后备人员可用，信息已经文档化，有关知识已经在团队中广泛进行了交流。此外，对那些人员充足的岗位，项目管理者还可以暂时重新调整资源（并重新调整项目进度），从而使得新加入团队的人员能够“赶上进度”。同时，应该要求那些将要离开的人员停止所有的工作，并在最后几星期进入“知识交接模式”。比如，准备录制视频知识，建立“注释文档或 Wiki”，或者与仍留在团队中的成员进行交流。

值得注意的是，RMMM 步骤会导致额外的项目成本。例如，花时间给每个关键技术人员配备“后备人员”得承担费用。因此，风险管理的另一个任务就是评估什么情况下由 RMMM 步骤所产生的效益高于实现这些步骤所需的成本。通常，项目管理者要进行典型的成本/效益分析。如果频繁的人员变动风险的缓解步骤经评估将会增加 15% 的项目成本和工期，而主要的成本因素是“配备后备人员”，则管理者可能决定不执行这一步骤。另一方面，如果风险缓解步骤经预测仅增加 5% 的项目成本和 3% 的工期，则管理者极有可能将这一步骤付诸实现。

建议 如果某特定风险的 RE 小于其风险缓解的成本，则不用试图缓解该风险，而是继续监测。

对于大型项目，可以识别出 30 或 40 种风险。如果为每一个风险制定 3 ~ 7 个风险缓解步骤，则风险管理本身就可能变成一个“项目”！因此，项目管理者可以将 Pareto 的 80-20 法则用于软件风险上。经验表明，整个项目 80% 的风险（即可能导致项目失败的 80% 的潜在因素）可能是由只占 20% 的已经识别出的风险所引发。早期风险分析步骤中所做的工作能够帮助项目管理者确定哪些风险在这 20% 中（如导致高风险显露度的风险）。因此，某些

已经识别、评估和预测过的风险可能并不被纳入 RMMM 计划之中——这些风险不属于那关键的 20%（具有最高项目优先级的风险）。

风险并不仅限于软件项目本身。在软件已经成功开发并交付给客户之后，仍有可能发生风险。这些风险一般与该领域中的软件缺陷相关。

软件安全和灾难分析（software safety and hazard analysis，例如 [Dun02]、[Her00]、[Lev95]）是一种软件质量保证活动（第 21 章），主要用来识别和评估可能对软件产生负面影响并促使整个系统失效的潜在灾难。如果能够在软件工程过程的早期阶段识别灾难，就可以使用某些软件设计特性来消除或控制这些潜在的灾难。

35.7 RMMM 计划

风险管理策略可以包含在软件项目计划中，也可以将风险管理步骤组织成一个独立的风险缓解、监测和管理计划（RMMM 计划）。RMMM 计划将所有风险分析工作文档化，项目管理者还可将其作为整个项目计划的一部分。

有些软件团队并不建立正式的 RMMM 文档，而是将每个风险分别使用风险信息表单（Risk Information Sheet, RIS）[Wil97] 进行文档化。在大多数情况下，RIS 采用数据库系统进行维护，这样容易完成创建、信息输入、优先级排序、查找以及其他分析。RIS 的格式如图 35-4 所示。

风险信息表单			
风险标识号: P02-4-32	日期: 5/9/09	概率: 80%	影响: 高
描述 事实上，计划可复用的软件构件中只有 70% 将集成到应用中，其他功能必须定制开发。			
细化 / 环境 子条件 1: 某些可复用构件是由第三方开发的，没有其内部设计标准的相关资料。 子条件 2: 构件接口的设计标准尚未确定，有可能和某些现有的软件可复用构件不一致。 子条件 3: 某些可复用构件是采用不支持目标环境的语言开发的。			
缓解 / 监测 1. 与第三方交流以确定其与设计标准的符合程度。 2. 强调接口标准的完整性，在确定接口协议时应考虑构件的结构。 3. 检查并确定属于子条件 3 的构件数量，检查并确定是否能够获得语言支持。			
管理 / 应急计划 / 触发 RE 的计算结果为 \$20200。在项目应急计划中分配这些费用。修订进度表，假定必须定制开发 18 个附加构件。据此分配人员。 触发: 缓解步骤自 7/1/09 起没有效果。			
当前状态 5/12/09: 缓解步骤启动。			
创建者: D. Gagne		受托者: B. Laster	

图 35-4 风险信息表单 [Wil97]

建立了 RMMM 计划，而且项目已经启动之后，风险缓解及监测步骤也就开始了。正如前面讨论过的，风险缓解是一种问题规避活动，而风险监测则是一种项目跟踪活动，这种监测活动有三个主要目的：（1）评估所预测的风险是否真正发生了；（2）保证正确地实施了各

风险的缓解步骤；(3) 收集能够用于今后风险分析的信息。在很多情况下，项目中发生的问题可以追溯到不止一个风险，所以风险监测的另一个任务就是试图找到“起源”(在整个项目中是哪些风险引起了哪些问题)。

软件工具 风险管理

[目标] 风险管理工具的目的是辅助项目团队识别风险，评估风险的影响及发生的概率，并在整个软件项目过程中跟踪风险。

[机制] 通常，风险管理工具能够提供典型的项目和商业风险列表来辅助识别一般风险；能够提供检查表或其他“接口”技术来辅助识别项目特定的风险；能够指定每一个风险发生的概率及影响；支持风险缓解策略；能够生成多种不同的风险相关报告。

[代表性工具]^①

- **@Risk**。由 Palisade Corporation (www.palisade.com) 开发，是一个利用蒙特卡罗 (Monte Carlo) 模拟法来驱动分析机的一般风险分析工具。

- **Riskman**。由 ABS Consulting (www.abs-consulting.com/riskmansoftware/index.html) 发布，是能够识别项目相关风险的一个风险评估专家系统。
- **Risk Radar**。由 SPMN (www.spmn.com) 开发，能够辅助项目管理者识别和管理项目风险。
- **ARM**。由 Deltek (www.deltek.com) 开发，这是一个基于 Web 的工具，可使供应商、客户及地处异地的项目团队共享必要的风险认知。
- **X:PRIMER**。由 GrafP Technologies (www.grafp.com) 开发，是一个通用的 Web 工具，可预测项目中什么可能出错，并能够识别出潜在错误的根本原因并制定有效的应对措施。

35.8 小结

对软件项目期望很高时，一般都会进行风险分析。不过，即使进行这项工作，大多数软件项目管理者也都是非正式和表面上完成它。花在识别、分析和管理风险上的时间可以从多个方面得到回报：更加平稳的项目进展过程，较高的跟踪和控制项目的能力，在问题发生之前已经做了周密计划而产生的信心。

风险分析需要占用大量项目计划的工作量。识别、预测、评估、管理和监测都要花费时间，但这是值得的。引用中国 2500 多年前的军事家孙子的一句话：“知己知彼，百战不殆。”对于软件项目管理者而言，这个“彼”指的就是风险。

习题与思考题

- 35.1 举出 5 个其他领域的例子来说明与被动风险策略相关的问题。
- 35.2 简述“已知风险”和“可预测风险”之间的差别。
- 35.3 为 SEPA 站点上给出的每个风险条目检查表增加三个问题或主题。
- 35.4 你受命开发一个支持低成本的视频编辑系统的软件。该系统输入数字视频信号，将视频信息存

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自开发者注册为商标。

在磁盘上,然后允许用户对数字化的视频信息进行各种编辑,最后生成DVD或其他多媒体格式。对这类系统做一些调研,然后列出当你开始启动这类项目时,将面临的技术风险是什么。

- 35.5 假如你是某大型软件公司的项目经理,且受命领导一个团队开发下一代字处理软件,请给该项目提供一个风险表。
- 35.6 说明风险因素和风险驱动因子的差别。
- 35.7 为图 35-2 所描述的三个风险制定风险缓解策略及特定的风险缓解活动。
- 35.8 为图 35-2 所描述的三个风险制定风险监测策略及特定的风险监测活动。确保你所监测的风险因素可以确定风险正在变大或变小。
- 35.9 为图 35-2 所描述的三个风险开发风险管理策略和特定的风险管理活动。
- 35.10 细化图 35-2 中的三个风险,并为每个风险建立风险信息表单。
- 35.11 用 CTC 格式表示图 35-2 中的三个风险。
- 35.12 假定每个 LOC 的成本为 \$16, 概率为 60%, 重新计算 35.4.2 节中讨论的风险显露度。
- 35.13 你能否想到一种情况:一个高概率、高影响的风险并未纳入 RMMM 计划的考虑之中?
- 35.14 给出主要关注软件安全和灾难分析的 5 个软件应用领域。

792

扩展阅读与信息资源

近几十年来,软件风险管理方面的文献已有很多。Mun (《Modeling Risk》, 2nd ed., Wiley, 2010) 详细介绍了适用于软件项目的风险分析数学处理方法。Mulcahy (《Risk Management, Tricks of the Trade for Project Managers》2nd ed., RMC Publications, 2010)、Kendrick (《Identifying and Managing Project Risk》2nd ed., American Management Association, 2009)、Crohy 及其同事 (《The Essentials of Risk Management》, McGraw-Hill, 2006) 以及 Marrison (《The Fundamentals of Risk Measurement》, McGraw-Hill, 2002) 介绍了每个项目采用的各种有用的方法及工具。Jindal 及其同事 (《Risk Management in Software Engineering, Create Space in Dependent Publishing》, 2012) 讨论了作为系统开发一部分的嵌入式安全风险评估。

DeMarco 和 Lister 写了一本有趣而意义深刻的书 (《Dancing with Bears》, Dorset House, 2003), 该书可以指导软件管理者和开发者进行风险管理。Moynihan (《Coping with IT/IS Risk Management》, Springer-Verlag, 2002) 介绍了项目风险管理者所采用的实用方法。Royer (《Project Risk Management》, Management Concepts, 2002) 以及 Smith 和 Merritt (《Proactive Risk Management》, Productivity Press, 2002) 论述了项目管理的主动过程。Karolak 撰写了一本参考书 (《Software Engineering Risk Management》, Wiley, 2002), 书中介绍了一种便于使用的风险分析模型, 以及一些有价值的检查表和调查表软件包。

Capers Jones (《Assessment and Control of Software Risks》, Prentice-Hall, 1994) 对软件风险进行了详细讨论, 其中包含从数百个软件项目中收集的数据, Jones 定义了 60 个可能影响软件项目结果的风险因素。Boehm[Boe89] 给出了很好的调查表和检查表格式, 对风险识别具有重大作用。Charette[Cha89] 描述了风险分析方法的详细处理, 采用了概率论和统计技术来分析风险。Charette 的另一本书 (《Application Strategies for Risk Analysis》, McGraw-Hill, 1990) 从系统和软件工程的角度讨论风险, 并提出了实用的风险管理策略。Gilb (《Principles of Software Engineering Management》, Addison-Wesley, 1988) 提出了一组“原则”(通常是有趣的, 有时是深刻的), 可作为风险管理的有效指南。

Ewusi-Mensah (《Software Development Failures: Anatomy of Abandoned Projects》, MIT Press, 2003) 和 Yourdon (《Death March》, Prentice-Hall, 1997) 讨论了当灾难性软件项目风险发生时会给

软件团队带来什么后果。Bernstein (《Against the Gods》，Wiley, 1998) 描述了有趣的远古时期的风险历史。

美国卡内基·梅隆大学软件工程研究所 (SEI) 已经出版了很多关于风险分析和风险管理的详细报告和参考书。美国空军司令部手册 AFSCP 800-45[AFC88] 描述了风险识别及降低技术。《ACM Software Engineering Notes》每期都有题为“Risk to the Public”(编辑: P. G. Neumann) 的讨论, 如果你想知道最新和最好的软件恐怖故事, 这里就有。

网上有大量与软件风险管理相关的信息资源, 最新的风险管理参考文献见 SEPA 网站 www.mhhe.com/pressman 上的“software engineering resource”专题。

793
|
794

维护与再工程

要点浏览

概念: 想想任何一个你曾经觉得很好的技术产品。虽然你定期使用它,但是它正在逐渐老化,经常出故障,对它进行修复所花费的时间越来越长,已经使你不可忍受,同时,它也不再代表最新的技术。怎么办?有一段时间,你试图修复它、给它打补丁甚至扩展它的功能,这称为维护。但是,随着时间的推移,维护变得越来越困难,到了重新构造它的时候了。构造一个具有更好的功能、更好的性能、更好的可靠性以及更好维护的产品。这就是我们所说的再工程。

人员: 在组织层次上,维护由支持人员(软件工程组织的一部分)完成,再工程由业务专家(通常是咨询公司)完成。在软件层次上,再工程由软件工程师完成。

重要性: 我们生活在快速变化的世界中,业务功能的要求和支撑它们的信息技术的变化步伐给每一个商业组织带来了巨大的竞争压力,这就是为什么必须对软件进行持续的维护,并在适当的时候对其实施再工程,以跟上变化的步伐。

步骤: 维护改正缺陷,对软件进行适应

性修改以满足变化的环境,增强功能以满足客户不断进化的需求。在策略级,业务过程再工程(Business Process Reengineering, BPR)制定业务目标,识别和评估现有的业务过程,并对业务过程进行修订,以更好地满足当前的业务目标。软件再工程过程包括库存目录分析、文档重构、逆向工程、程序和数据重构,以及正向工程。这些活动的意图是创建具有更高质量和更易于维护的现有程序的新版本。

工作产品: 产生一系列可维护的和再工程工作产品(例如,用例、分析模型和设计模型、测试规程),最终产品是升级的软件。

质量保证措施: 再工程中使用的 SQA(软件质量保证)实践与应用于各个软件工程的 SQA 实践相同——用正式技术评审来评估分析模型和设计模型,用专门的评审考查业务适用性和兼容性,用测试来发现内容、功能性和互操作性方面的错误。

不管计算机软件的应用领域、规模或复杂性如何,计算机软件都将随时间而不断演化,因为变更驱动着这个过程。对于计算机软件,在很多情况下都会发生变更:当进行纠错时会发生变更;当修改某个软件以适应新环境时会发生变更;当客户要求新的特性或新功能时会发生变更;将应用再工程以适应现代环境时也会发生变更。在过去的 40 年中, Manny Lehman [如 Leh97a] 及其同事已经对行业级的软件和系统进行了详细分析,目的是为了得到软件演化的统一理论(unified theory for software evolution)。虽然本书没有详细介绍这项工

作，但是其所推导出的基本规律仍值得关注 [Leh97b]:

持续变更法则 (1974): 由于软件是在真实世界的计算环境中实现的，因此将随着时间不断演化 (称为 E 型系统)，所以必须持续修改软件，否则这些软件将变得越来越令人不满。

复杂性增长法则 (1974): 随着 E 型系统的不断演化，系统的复杂性也随之增长，除非采取措施使系统保持或降低复杂性。

自调节法则 (1974): E 型系统的演化过程可以自动调节产品分布和过程测量以接近正常状态。

组织稳定性守恒法则 (1980): 演化中的 E 型系统的平均有效全局活动率在整个产品的生命期内是恒定的。

熟悉度守恒法则 (1980): 随着 E 型系统的不断演化，与该系统相关的所有人员，(如开发人员、销售人员、用户) 都必须始终掌握系统的内容和行为，以使得演化过程令人满意。过度的增长会削弱对其内容和行为的掌握程度。因此，在系统的演化过程中，平均增长值是恒定的。

持续增长法则 (1980): 在 E 型系统的整个生命期内，其功能内容必须持续增长以满足用户需求。

质量下降法则 (1996): 如果 E 型系统没有严格地进行维护，也没有随着操作环境的改变做适应性修改，那么 E 型系统的质量将有所下降。

反馈系统法则 (1996): E 型演化过程由多层次、多循环、多代理的反馈系统组成，而且，要想在任何合理的基础上达到有意义的改进就必须这样进行处理。

Lehman 及其同事定义的这些法则是软件工程现实中的固有部分。本章将讨论软件维护所面临的挑战以及延长遗留系统的有效生命周期所需要的再工程活动。

关键概念

业务过程再工程
文档重构
正向工程
库存目录分析
可维护性
重构
代码
数据
逆向工程
数据
处理
用户界面
软件维护
软件再工程
可支持性

提问 为什么遗

留系统会随着时间的推移而发生进化?

796

36.1 软件维护

软件维护几乎是在软件交付给最终用户后就立即开始。软件交付给最终用户后，没几天功夫，缺陷报告就有可能送到软件工程组织；没几周功夫，某类用户就可能会提出必须修改软件以适应他们所处环境的特殊要求；没几个月功夫，另一个公司在软件发布时认为他们与这个软件毫不相干，但现在意识到该软件可能会给他们带来意想不到的好处，因此他们需要做些改进，使软件可以用于他们的环境。

软件维护所面临的挑战已经开始。软件工程组织面临着不断增长的代码错误修改任务、适应性修改请求及彻底的增强，这些都必须进行计划、安排进度并最终完成。用不了多久，维护队列就已经很长，这意味着修正工作可能会用尽现有的资源。随着时间的推移，软件工程组织会发现自己花在维护现有程序上的资金和时间远比创建新的应用多得多。事实上，软件组织将全部资源的 60% ~ 70% 花费在软件维护上是很常见的。

有人可能会问：为什么需要这么多的维护？为什么要花费这样多的工作量来进行维护？Osborne 和 Chikofsky[Os90] 给出了部分答案：

我们现在使用的很多软件已有 10 ~ 15 年的历史。即使这些程序是采用当时最好的设计和编码技术开发的 (大多数并不是这样)，但当时主要关注的是程序规模和存储空间。这些软件后来被移植到新的平台上，根据机器和操作系统技术方面的变化对其进行了调整，为满足新的用户要求对其进行了改进——所有这些并没有对整个体系结构给予足够关注，其结果

是，我们现在仍在运行的软件系统的设计结构、编码、逻辑及文档都很差……

软件维护问题的另一个原因是软件人员变动。最初从事开发工作的软件团队（或个人）可能已经不在。更糟糕的是，后来的软件人员已经修改了系统，最后也离开了。目前，已经没有人直接了解这个遗留系统了。

就像在第 29 章中讲到的，所有软件工作都普遍在变更之中进行。开发计算机系统时，变更是不可避免的。因此，软件组织一定要建立评估、控制及进行修改的机制。

本书从始至终都一直强调弄清楚问题（分析）和给出结构明确的解决方案（设计）的重要性。实际上，本书第二部分就主要讨论了这两个软件工程活动的机制，第三部分重点介绍了能够保证软件组织正确完成这两个软件工程活动所需要的技术。分析和设计都可以达到一个重要的软件特性，即可维护性。本质上，可维护性（maintainability）是一个定性指标^①，它表明对现有软件进行改正、适应或增强的容易程度。软件工程所涉及的大部分内容都是构建能够表现出高可维护性的系统。

引述 程序可维护性和程序可理解性是相关的概念：程序越难于理解，也就越难于维护。

Gerald Berns

但什么是可维护性？可维护的软件表现为有效的模块化（第 12 章），它采用易于理解的设计模式（第 16 章），构建时采用了明确定义的编码标准及约定，编写的源代码能够自身文档化且易于理解。它应用了大量质量保证技术（本书第三部分），在软件交付之前就已经找出了潜在的维护问题。创建它的软件工程师已经意识到：实施变更时他们可能已经离开了，因此，软件的设计与实现必须对实施变更的人员“有帮助”。

797

36.2 软件可支持性

为了有效支持行业级软件，一个组织（或其指定的人员）必须具有完成修正、改写及改进的能力，而修正、改写以及改进是维护活动的一部分。除此之外，在软件的整个生命周期内，该组织还必须提供运行支持、最终用户支持以及再工程活动等重要的支持活动。软件可支持性（software supportability）的合理定义如下：

……在软件系统的整个产品生命周期内支持软件系统的能力。这意味着不仅要满足所有必要的要求或需求，而且还要能够供应装置、支持基础设施、附加软件、工具、人力或所需要的任何资源以维护软件的运行并满足软件功能 [SSO08]。

本质上，在软件过程中进行分析和设计时就应考虑作为质量因素之一的可支持性。可支持性应该在需求模型（或规格说明）中进行处理，并且要在设计的演化过程中以及构建开始时认真考虑。

例如，本书前面已经讨论过构件级和代码级的“防错”（antibug）软件需求。当在运行环境中出现缺陷时（没有错误，但会有缺陷），该软件应该具有协助支持人员进行处理的工具。此外，支持人员应该能够访问到记录已经出现的所有缺陷的数据库，获得各个缺陷的特征、起因以及补救措施。这就使得支持人员能够考察“类似的”缺陷，从而提供更快的诊断及纠错手段。

虽然在应用中出现缺陷属于关键性的支持问题，但是，可支持性还需要提供相应的资源以支持日复一日的最终用户问题。最终用户支持人员的工作就是回答用户对应用的安装、运行以及使用等方面的疑问。

网络资源 有关软件可支持性的
大量可下载的文
献见 www.softwar-supportability.org/Downloads.html。

① 有些定量测量可以提供可维护性的间接指标（例如 [Sch99]、[SEI02]）。

36.3 再工程

Michael Hammer 在《Harvard Business Review》上撰写的研讨论文 [Ham90] 为业务过程和计算方面的管理变革奠定了基础：

现在是该停止铺设牛道的时候了。不要将过时的工艺过程嵌入到硅片和软件中，我们应该将它们删除，并从头开始。应该对我们的业务进行“再工程”：采用现代信息技术的优势重新设计业务过程，以获得性能的极大改善。

对再工程的大肆宣传已经减弱，但是这个过程本身仍在大大小小的公司中继续进行。业务再工程和软件工程间的关系开始进入“系统视线”中。

当管理者为了获得更高的效率和竞争力而修改业务规则时，软件也必须保持同步。在某些情况下，这意味着需要构建大量新的计算机系统^①，但是，在多数情况下，只需要对现有的应用进行修改或重构。

在下面的章节中将以自顶向下的方式考察再工程，从业务过程再工程的概述开始，进而对软件再工程中发生的技术活动进行更详细的讨论。

引述 用昨天的思考方法面对明天就是在停顿中想象生活。

James Bell

36.4 业务过程再工程

业务过程再工程（Business Process Reengineering, BPR）远远超出了信息技术和软件工程的范畴。有很多关于 BPR 的定义（大多数有点抽象），刊登于《财富》杂志 [Ste93] 的定义为：“搜寻并实现业务过程中的根本性改变，以取得突破性成果。”但是，如何进行搜寻？如何获得实现？更重要的是，我们如何能够保证所建议的“根本性改变”确实能够取得“突破性成果”，而又不会造成组织的混乱？

关键点 虽然软件再工程是以更好的、更易于维护的软件替换现有的软件功能，但是 BPR 经常能够得到新的软件功能。

36.4.1 业务过程

业务过程是指“执行一组逻辑相关的任务以获得定义明确的业务结果”[Dav90]。业务过程将人员、设备、材料资源以及业务规程综合在一起以产生特定的结果。业务过程的例子有：设计新产品、购买服务和支持、雇用新员工，以及向供应商付费。每种业务过程都需要一组任务，而且要在业务中利用不同的资源。

每个业务过程都有一个指定的客户——接收业务过程结果（例如，想法、报告、设计、服务、产品）的个人或小组。此外，业务过程一般会跨越组织边界，需要来自不同组织的小组共同参与定义过程的“逻辑相关的任务”。

建议 作为软件工程师，你的工作位于业务层次的底部。但是，要确保有人已对上层的工作进行过认真考虑。如果不是这样，你的工作便处在危险之中。

各个系统实际上都是由多个子系统构成的层次结构。业务也不例外，总业务可以按照下面的方式进行分解：

业务→业务系统→业务过程→业务子过程

每一个业务系统（也称为业务功能）可以由一个或多个业务过程组成，而每个业务过程又可定义为一组子过程。

BPR 可以应用于层次结构的任意级别，但是，随着 BPR 范围的扩大（即在层次中向上

① 基于 Web 的应用及系统的爆炸性发展就预示了这种趋势。

移动), 与 BPR 关联的风险将急剧增长。正因为如此, 大多数 BPR 工作只着重于单个过程或子过程。

36.4.2 BPR 模型

和大多数工程活动一样, 业务过程再工程也是迭代的。所有的业务目标及为达到这些目标所采用的过程都必须能够适应不断变化的业务环境。因此, BPR 没有开始和结束——它是一个演化的过程。图 36-1 描述了一个业务过程再工程模型, 该模型定义了以下 6 项活动。

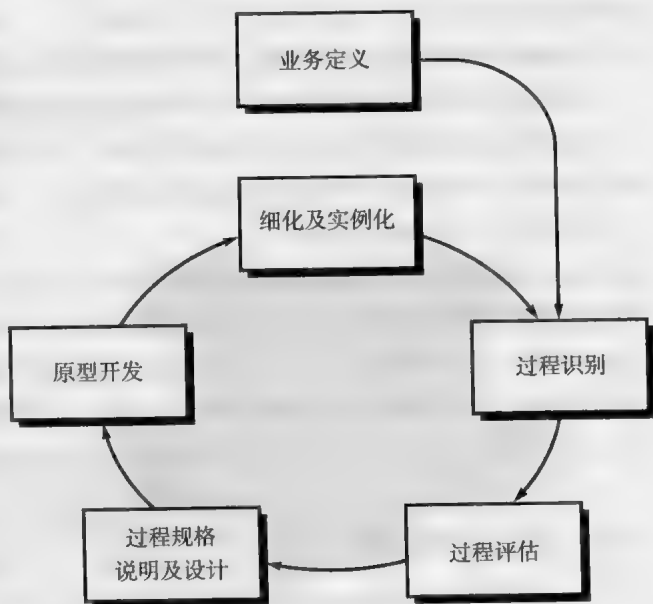


图 36-1 BPR 模型

- 1. 业务定义。**在 4 个关键的驱动因子范畴内制定业务目标, 这 4 个关键的驱动因子是降低成本、缩短时间、提高质量、人力开发以及授权。可以在业务级或针对业务的某个特定部分来制定业务目标。
- 2. 过程识别。**识别对业务定义中所制定的业务目标起关键作用的所有过程, 然后可以按照重要性、变更的要求或以任何适用于再工程活动的方式将这些过程排序。
- 3. 过程评估。**对现有过程进行透彻的分析和测量。确定所有的过程任务, 说明这些过程任务花费的成本和时间, 并且明确质量 / 性能问题。
- 4. 过程规格说明及设计。**根据从上面 3 个 BPR 活动中获得的信息, 为每个即将被重新设计的过程准备用例 (第 8 章和第 9 章)。在 BPR 范畴内, 这些用例描述了将某些结果传递给客户的场景。将这些用例作为过程的规格说明, 为过程设计一组新任务。
- 5. 原型开发。**在将重新设计的业务过程完全集成到整个业务之前, 必须将其原型化。这个活动可以“测试”重新设计的业务过程, 有利于实现细化。
- 6. 细化及实例化。**根据来自原型的反馈信息对业务过程进行细化, 然后在业务系统中实例化。

上述 BPR 活动有时是和工作流分析工具联合使用的, 使用这些工具的目的是建立现有工作流的模型, 以便对现有的过程进行更好的分析。

引述 只要能够
在新事物中展示出
往昔的东西, 我
们就会感到欣慰。
F. W. Nietzsche

软件工具 业务过程再工程

[目标] BPR 工具支持对现有业务过程的分析与估算, 以及新业务过程的规格说明与设计。

[机制] 工具的机制各异。一般情况下, BPR 工具允许业务分析员对现有的业务过程建模, 主要用于评定工作流的效率

缺陷或功能问题。一旦识别出存在的问题,这些工具还允许分析员开发原型和模拟修订的业务过程。

[代表性工具]^①

- ExtendSim。由 ImagineThat 公司 (www.imaginethatinc.com) 开发,是对现有过程建模及探索新过程的一种模拟工具。Extend 提供了全面的“如果……就……”(what if)能力,使得业务分析能够探索不同的过程场景。
- Metastrom BPM。由 OpenText (<http://bps.opentext.com/>) 开发,支持手动和自动过程的业务流程管理。
- IceTools。由 Blue Ice ([http://www.Iceto-](http://www.Iceto-ols.com/home.html)

[ols.com/home.html](http://www.Iceto-ols.com/home.html)) 开发,这是一组适用于微软 Office 及微软 Visio 的 BPR 模板。

- OMNIBUS。由 Kovair (<http://www.kovair.com>) 开发,是组织模拟过程工作流的一种工具(IT工作流)。
- ProcessMaker。开源工作流套件,由 Colosa(<http://www.processmaker.com/>) 开发,包含一套工作流建模、仿真和调度的工具。

可链接到 BPR 工具的有用列表见 www.opfro.org/index.html?Components/Producers/Tools/BusinessProcessReengineeringTools.html~Contents。

801

36.5 软件再工程

类似这样的情形实在很普遍:某应用已经为公司的业务需要服务了 10 年或 15 年,在此期间,已经对它进行了多次纠错、适应性修改及增强。人们怀着极好的愿望从事这项工作,但是,好的软件工程实践总是被抛在一边(由于其他方面的压力)。现在,应用已经处于不稳定的状态,虽然它仍在工作,但每次维护都会产生预料不到的、严重的副作用。然而,这个应用必须继续使用,怎么办?

不可维护的软件并不是什么新问题。事实上,对软件再工程的强调源于近半个世纪以来软件维护问题的不断升温。

36.5.1 软件再工程过程模型

再工程要花费时间,消耗大量的资金并占用资源,而这些花费本可用于当前关注的事情上。因此,再工程不是在几个月或甚至几年内可以完成的。信息系统的再工程将是消耗信息技术资源长达多年的活动,因此每个组织都需要注重有实效的软件再工程策略。

再工程过程模型提供了一个可行的策略,我们将在本节后面讨论该模型,下面首先讨论一些基本原则。

再工程是一项重构活动。为了更好地理解再工程,我们考虑与再工程相似的“重建一所房子”的活动。考虑如下情况:假定你在另一个州购买了一所房子。你还没有亲眼看到房子,就以令人吃惊的低价格买下了。在可能需要彻底重建的情况下,你将如何进行重建?

- 在开始重建前,先检查一下房子似乎是合理的做法。为了确定它是否确实需要重建,你(或职业检查员)可以列出一组标准,使得检查工作能系统地进行。

网络资源 有关软件再工程的信息源见 reengineer.org。

① 这里提到的工具只是此类工具的例子,并不代表本书支持采用这些工具。在大多数情况下,工具名称被各自的开发者注册为商标。

- 在拆掉并重建整座房子前，确认其结构是不是牢固的。如果该房子结构良好，则可能只需要“改造”(remodel)，而不是重建（花更低的成本、更少的时间）。
- 在开始重建前，确保你已经了解房子最初是如何建造的。看一看墙内部，弄清楚布线、管道以及内部结构。即使你不顾所有这些，详细了解原房屋对你开始建造也一定是有帮助的。
- 如果开始重建，应该只使用最现代、最耐久的材料。现在看来可能会贵一些，但是，这样做会使你避免将来昂贵而耗时的维护。
- 如果决定重建，一定要采用严格的方式，使用现在及将来都将获得高质量的做法。

802

虽然上面的原则针对的是房子的重建，但它们也同样适用于计算机系统和计算机应用的再工程。

为了贯彻这些原则，可以使用图 36-2 所示的软件再工程过程模型，它定义了 6 类活动。在某些情况下，这些活动以线性顺序出现，但并不总是这样。例如，有可能在文档重构开始前，必须先进行逆向工程（弄清楚程序的内部工作原理）。

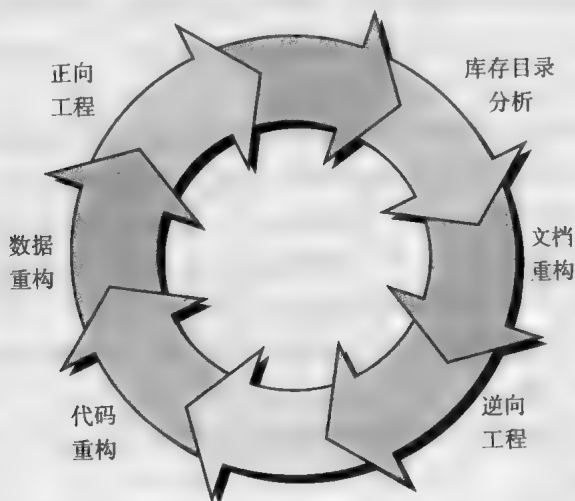


图 36-2 软件再工程过程模型

36.5.2 软件再工程活动

图 36-2 所示的再工程范型是一个循环模型。这意味着该范型中的每项活动均可能重复出现。对任意一个特定的循环，其过程可以在任意一项活动之后终止。

库存目录分析。各软件组织都应该保存所有应用的库存目录。该目录可能仅仅是一个电子表格模型，其中为每个常用的应用提供了详细的描述（例如，规模、年限、业务重要程度）。按照业务重要程度、寿命、当前可维护性和可支持性以及其它本地重要性准则对这些信息进行排序，可以选出再工程的应用，然后为这些应用的再工程工作分配资源。

值得注意的是：应该对库存目录进行定期分析。因为应用的状态（例如，业务重要程度）可能随时间发生变化，从而会使再工程的优先级发生变化。

文档重构。拙劣的文档是很多遗留系统的特点。但是，对此能做些什么呢？如何进行选

建议 如果时间及资源紧缺，则可以考虑将 Pareto 原理应用到将要实施再工程的软件中。将再工程过程应用到存在 80% 问题的 20% 的软件中。

803

择？在某些情况下，当没有素材时，建立文档是非常耗费时间的。如果系统正常运作，可以选择保持现状，然而在发生变更时，则必须进行文档化。如果系统是业务关键的，而且必须完全重构文档，即使出现这种情况，也最好是设法将文档精简到最少。软件组织必须针对不同的情形选择最适合的方法。

建议 只建立理解软件所需要的文档，一页都不要多写。

逆向工程。软件的逆向工程是分析程序、在高于源代码的抽象层次上表示程序的过程。逆向工程是一个设计恢复（design recovery）过程。逆向工程工具从现有的程序中抽取数据、体系结构和过程的设计信息。

代码重构。最常见的再工程（实际上，在这里使用术语再工程是有疑问的）类型就是代码重构^①（code restructuring）。有些遗留系统具有相对可靠的程序体系结构，但是，个别模块的编码方式使得程序难于理解、测试和维护。在这样的情形下，可以对可疑模块内的代码进行重构。

要实现代码重构，可以使用重构工具去分析源代码，将与结构化程序设计概念相违背的部分标注出来，然后对代码进行重构（此工作可以自动进行）或者用更现代的程序设计语言重新编写。对生成的重构代码进行评审和测试，以确保没有引入不规则的代码，并更新内部的代码文档。

数据重构。数据体系结构差的程序将难于进行适应性修改和增强。事实上，对大部分应用来说，数据体系结构比源代码本身对程序的长期生存力影响更大。

与抽象层次较低的代码重构不同，数据重构是一种全范围的再工程活动。在大多数情况下，数据重构开始于逆向工程活动。对当前的数据体系结构进行分解，并定义必要的数据模型，标识数据对象和属性，并对现有的数据结构进行质量评审。

当数据结构较差时（例如，通常采用平面文件实现，而关系型方法可以大大地简化处理），应该对数据进行再工程。

由于数据体系结构对程序体系结构及其算法有很强影响，所以对数据的变更总会导致体系结构或代码层的变更。

正向工程。在理想情况下，可以使用自动的“再工程引擎”来重建应用。将旧程序输入引擎，经过分析、重构，然后重新生成能够表现出最好的软件质量的程序。短期内，这样的“引擎”还不可能出现，但是，有的厂商已经开发了针对特定应用领域（例如，用特定数据库系统实现的应用）的一些工具，能够实现一部分功能。更重要的是，这些再工程工具正在变得越来越成熟。

正向工程不仅能够从现有软件恢复设计信息，而且还能够使用这些信息去改变或重构现有系统，以提高其整体质量。大多数情况下，实施了再工程的软件可以重新实现现有系统的功能，并且还能够加入新功能和提高整体性能。

36.6 逆向工程

逆向工程假想有一个“魔术通道”，在通道的一端输入随意设计的、无文档的源程序文件，另一端出来的就是计算机程序的完整设计描述（以及完整文档）。不幸的是，这样的魔

^① 代码重构具有“重构”（refactoring）的某些成分，重构是一个设计概念，在第12章中介绍了这个概念，本书的其他地方也进行了讨论。

术通道并不存在。逆向工程可以从源程序中抽取设计信息，但是，抽象的层次、文档的完备性、工具与分析人员协同工作的程度、过程的方向性却是高度可变的。

逆向工程过程以及用于实现逆向工程过程的工具的抽象层次 (abstraction level) 是指可以从源代码中抽取出来的设计信息的精密程度。理想情况下，抽象层次应该尽可能高，即逆向工程过程应该能够推导出过程的设计表示 (低层抽象)，程序和数据结构信息 (稍高层次的抽象)，对象模型、数据和控制流模型 (相对高层的抽象) 以及实体关系模型 (高层抽象)。随着抽象层次的增高，软件工程师能够得到更有助于理解程序的信息。

逆向工程过程的完备性 (completeness) 是指在某一抽象层次上提供信息的详细程度。在大多数情况下，随着抽象层次的增高，完备性就会降低。例如，给定源代码列表，要得到完整的过程设计表示是比较容易的。虽然也可以推导出简单的体系结构设计表示，但要得到 UML 图或模型的完整集合却困难得多。

完备性的改善与做逆向工程的人员所完成的分析量成正比。交互性 (interactivity) 是指为了建立一个有效的逆向工程过程，人员与自动工具“结合”的程度。在大多数情况下，随着抽象层次的增高，交互性必定增加，否则完备性将受到损害。

如果逆向工程过程的方向性 (directionality) 是单向的，则从源程序中抽取的所有信息都将提供给软件工程师，然后他们可以在任何维护活动中使用这些信息。如果方向性是双向的，则需要将这些信息输入到再工程工具，以试图重构或重新生成旧程序。

逆向工程过程如图 36-3 所示，在逆向工程活动可以开始前，要重构无结构的 (“脏的”) 源代码 (36.5.1 节)，使得它仅包含结构化程序设计结构[⊖]。这样使得源代码更容易理解，并为所有后续的逆向工程活动奠定基础。

逆向工程的核心活动是抽取抽象 (extract abstraction)。工程师必须评估旧程序，并从 (通常是无文档的) 源代码中抽取有意义的规格说明，包括要执行的处理、要使用的用户界面以及所采用的程序数据结构或数据库。

36.6.1 理解数据的逆向工程

数据的逆向工程可以发生在不同的抽象层次，并且通常是第一项逆向工程任务。在程序层，作为整体再工程工作的一部分，必须对内部程序的数据结构进行逆向工程。在系统层，通常要将全局数据结构 (例如，文件、数据库) 实施再工程以符合新的数据库管理范型 (例如，从平面文件转移到关系数据库系统或面向对象数据库系统)。对现有全局数据结构的逆向工程为引入新的系统范围的数据库奠定了基础。

内部数据结构。针对内部程序数据的逆向工程技术着重于对象的类定义。对程序代码进行检查，组合相关的程序变量以完成类的定义。在很多情况下，代码中的数据组织标识

关键点 逆向工程必定要涉及的问题是抽象层次、完备性和方向性。

网络资源 “设计恢复及程序理解”的有用资源见 www.softpanorama.net/SE/reverse_engineering_links.shtml。

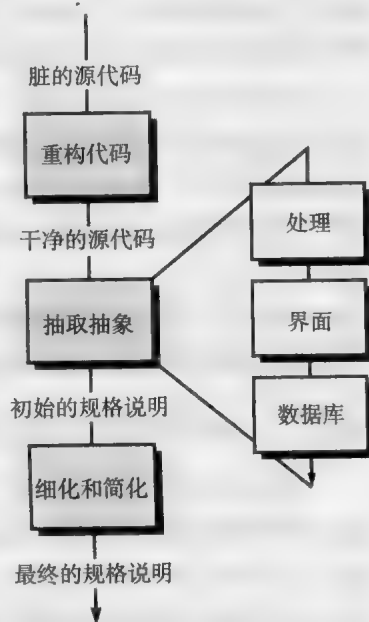


图 36-3 逆向工程过程

⊖ 可使用重构引擎 (restructuring engine) 对代码进行重构，重构引擎是一种可以重构源代码的工具。

了抽象数据类型,例如,记录结构、文件、列表和其他数据结构通常都给出了类的最初指示。

数据库结构。不考虑其逻辑组织及物理结构,数据库允许定义数据对象,并支持在对象间建立关系的方法。因此,当要将一种数据库模式实施再工程以使之成为另一种数据库模式时,需要清楚现有对象及它们之间的关系。

为了对新数据模型实施再工程,首先要定义现有数据模型,可用下面的步骤 [Pre94]: (1) 构造初始的对象模型; (2) 确定候选关键字 (考察每一个属性,以确定其是否被用来指向另外的记录或另一张表,充当指针的那些属性就是候选关键字); (3) 细化实验性的类; (4) 定义一般化关系; (5) 使用与 CRC 方法相似的技术找出关联关系。一旦知道了在前面步骤中所定义的信息,则可以通过一系列转换 [Pre94] 将旧的数据库结构映射到新的数据库结构。

36.6.2 理解处理的逆向工程

处理的逆向工程开始于试图弄清楚并抽取出源代码所表示的过程抽象。为了弄清楚过程抽象,需要在不同的抽象级别分析代码:系统级、程序级、构件级、模式级和语句级。

在进行更详细的逆向工程前,必须弄清楚整个应用的整体功能。这项工作确定了进一步分析的范围,并对更大系统中应用间的互操作问题有了深入的理解。构成应用的每一个程序代表了在更高详细层次上的功能抽象,可以用结构图表示这些功能抽象间的交互作用。每一个构件实现某个子功能,同时也表示某个定义的过程抽象,所以要对每个构件的处理进行描述。在某些情况下,系统、程序和构件的规格说明已经存在,若是这种情况,则要对这些规格说明进行评审,以确认是否与现有代码相符[⊖]。

当考虑构件中的代码时,事情变得更复杂。工程师需找到表示通用过程模式的代码段。几乎在每个构件中都是由一个代码段准备 (在模块内) 要处理的数据,由另一个不同的代码段完成处理工作,然后再由另一个代码段准备构件要输出的处理结果。在每个代码段中,工程师都可能遇到更小的模式,例如,数据确认和范围检查经常出现在为处理准备数据的代码段中。

对于大型系统,通常采用半自动方法完成逆向工程。使用自动化工具帮助软件工程师弄清楚现有代码的语义,然后将该过程的结果传递给重构和正向工程工具以完成再工程过程。

36.6.3 用户界面的逆向工程

高级图形用户界面 (GUI) 对于计算机产品和各类计算机系统来说都是不可缺少的,因此,用户界面的重新开发已经成为最常见的再工程活动类型之一。但是,在重建用户界面之前,应该先进行逆向工程。

为了能够完全弄清楚现有的用户界面,必须详细说明界面的结构和行为。Merlo 及其同事 [Mer93] 提出了在用户界面 (UI) 的逆向工程开始前必须回答的三个基本问题:

建议 在某些情况下,第一个再工程活动是 UML 类图。

建议 针对传统软件的数据,逆向工程方法遵循类似的步骤: (1) 建造数据模型; (2) 标识数据对象的属性; (3) 确定关系。

引述 试图理解某一事物的激情,就像对音乐的激情一样,在孩子中间是常见的,但是,后来在大多数人中却消失了。

Albert Einstein

807

⊖ 通常,在程序生命历史早期编写的规格说明从未更新过。随着代码的修改,代码将不再与规格说明相符。

- 界面必须处理的基本动作（例如，击键和点击鼠标）是什么？
- 系统对这些动作的行为反应的简要描述是什么？
- “替代者”意味着什么？或更确切地说，在这里，有哪些界面的等价概念是相关的？

提问 如何弄清楚现有用户界面是怎样工作的？

对于前两个问题，行为建模符号（第 11 章）可以提供求解方法，创建行为模型所必需的信息多数可以通过观察现有界面的外部表现而得到，但是，还有一些信息必须从代码中抽取。

值得注意的是，替代的 GUI 可能并不是旧界面的精确镜像（实际上，它可能完全不同）。开发新的交互隐喻通常是值得的，例如，旧的 UI 要求用户提供比例因子（范围是 1 ~ 10），用来放大或缩小一幅图像，再工程后的 GUI 可能使用滚动条及鼠标来完成相同的功能。

808

软件工具

逆向工程

[目标] 帮助软件工程师弄清楚复杂程序的内部设计结构。

[机制] 在大多数情况下，逆向工程工具接收源代码作为输入，然后产生多种结构、过程、数据以及行为的设计表示。

[代表性工具][⊖]

- Imagix 4D。由 Imagix(www.imagix.com) 开发，通过逆向工程及编写源代码的文档，“帮助软件开发者理解复杂的或用 C 及 C++ 编写的遗留软件”。

- Understand。由 Scientific Toolworks, Inc. (www.scitools.com) 开发，能够分析 Ada、Fortran、C、C++、C#、PHP、HTML、JavaScript、Python 及 Java 程序，“对于逆向工程师，能够自动产生文档，计算代码度量，并帮助你理解、导航及维护源代码”。

逆向工程工具的完整列表可以在 <http://eclipse.org/gmt/modisco/telatedProjects.php> 找到。

36.7 重构

软件重构工作是要修改源代码和数据，使软件适应未来的变化。通常，重构并不修改总体程序结构，它倾向于关注单个模块的设计细节及模块中所定义的局部数据结构。如果重构扩展到模块边界之外，而且涉及软件体系结构，则重构变成了正向工程（36.8 节）。

当某应用的基本体系结构比较好，但技术的内部细节需要修改时，则需要对其进行重构。当软件的大部分是有用的，仅仅需要对部分模块和数据进行扩展性修改时，则启动重构活动[⊖]。

建议 虽然代码重构可以缓解与调试或小修改相关的问题，但是，它不是再工程。只有重构数据和体系结构，才能够取得真正的收益。

36.7.1 代码重构

进行代码重构（code restructuring）是为了生成与源程序具有相同功能但具有更高质量的设计。通常，代码重构技术（例如，Warnier 的逻辑简化技术 [War74]）都是利用布尔代数对程序逻辑进行建模，然后应用一系列变换规则来重构逻辑。其目的是采用“面条碗”式的

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

⊖ 扩展性重构和再开发有时很难区分，两者均属于再工程。

809 代码,并推导出遵从结构化程序设计原理(第19章)的过程设计。

建议将其他重构技术与再工程工具一同使用,资源交换图能够映射每个程序模块及其与其他模块间交换的资源(数据类型、过程和变量),通过创建资源流的表示,可以对程序体系结构进行重构,以达到模块间的最小耦合。

36.7.2 数据重构

在数据重构开始前,必须先进行称为源代码分析(analysis of source code)的逆向工程活动。评估所有包含了数据定义、文件描述、I/O以及接口描述的程序设计语言语句,目的是抽取数据项及对象,获取关于数据流的信息,以及弄清楚目前已实现的数据结构。有时也称该项活动为数据分析(data analysis)。

一旦完成了数据分析,就可以开始数据重设计(data redesign)。其最简单的形式为:通过数据记录标准化(data record standardization)步骤明确数据定义,从而使现有数据结构或文件格式中的数据项名或物理记录格式取得一致。另一种重设计形式称为数据名合理化(data name rationalization),这种重设计形式能够保证所有数据命名约定符合本地标准,并且当数据在系统内流动时可以忽略别名。

当重构超出标准化和合理化的范畴时,要对现有数据结构进行物理修改以使数据设计更为有效。这可能意味着从一种文件格式到另一种文件格式的转换,或在某些情况下,这也意味着从一种数据库类型到另一种数据库类型的转换。

软件工具 软件重构

[目标] 重构工具的目标是将旧的非结构化的计算机软件转化为现代程序设计语言及设计结构。

[机制] 通常,将源代码输入,然后将其变换为更好的结构化程序。在某些情况下,这一变换发生在同一种程序设计语言中。在有些情况下,能够将一种较老的程序设计语言变换成一种更加现代的语言。

[代表性工具]^①

- DMS Software Reengineering Toolkit。由 Semantic Design(www.semdesigns.com) 开发,提供了针对 COBOL、C/C++、Java、Fortran 90 及 VHDL 的多种重构能力。

- Clone Doctor。由 Semantic Designs, Inc. (www.semdesigns.com) 开发,能够分析和变换用 C、C++、Java、COBOL 或其他任何基于文本的计算机语言编写的程序。

- plusFORT。由 Polyhedron (www.polyhedron.com) 开发,是一套 Fortran 工具,能够将设计质量低的 Fortran 程序重构为现代的 Fortran 或 C 标准程序。

大量再工程和逆向工程的工具的链接可在 <http://www.comp.lancs.ac.uk/projects/Reengineering/Tools.html> 和 <http://www.fujaba.de/projects.html> 找到。

810

36.8 正向工程

程序的控制流在图形上相当于一碗意大利面,假设某程序有一个包含 2000 行语句的

^① 这里提到的工具只是此类工具的例子,并不代表本书支持采用这些工具。在大多数情况下,工具名称被各自的开发者注册为商标。

“模块”，在 290000 行源代码中几乎没有有意义的注释行，并且没有其他文档，如果该程序必须修改，以适应用户需求的变化，那么软件工程师有下列选择：

1. 努力地不断修改，与特定的设计和复杂的源代码搏斗，以实现必要的修改。
2. 试图理解程序更多的内部工作，努力使修改更有效。
3. 重设计、重编码并测试该软件需要修改的部分，对所有修改过的片段应用软件工程方法。
4. 完全地重设计、重编码并测试该程序，使用再工程工具来辅助理解现有的设计。

这里没有唯一的“正确”选择。当前遇到的情况可能要求我们做出第一种选择，即使更期望的是其他选择。

不要坐等维护请求，开发或维护组织应该从库存目录分析的结果中挑选出一个程序，该程序：（1）将在预先确定的年限内继续使用；（2）当前的使用是成功的；（3）很可能在不久的将来做较大的修改或增强。这样，我们就可以应用到上面的第 2～4 个选项。

乍一看，重新开发大型程序的现有可用版本似乎是相当浪费的，不过在做出判断之前，应该考虑如下几点：维护一行源代码的成本可能是该行代码初始开发成本的 20～40 倍。采用现代设计概念重新设计软件体系结构（程序或数据结构）可以大大地便于未来的维护。由于软件的原型已经存在，因此开发生产率将远高于平均水平。用户现在已对该软件有使用经验，因此，可以很容易地确定新的需求和变更的方向。再工程的自动化工具可以使部分工作简化。预防性维护的完成将产生完整的软件配置（文档、程序和数据）。

大规模的内部软件开发（例如，一个大型消费品公司的业务系统软件开发小组）可能在其职责范围内有 500～2000 个产品程序，可以根据重要程度对这些程序进行优先级排序，然后对所选出的要进行正向工程的程序进行评审。

在大多数情况下，正向工程并不仅仅是创建某旧程序的现代等价物，而是将新的用户和技术需求集成到再工程中，重新开发的程序可以扩展原有应用的能力。

36.8.1 客户 / 服务器体系结构的正向工程

在过去的几十年中，集中式计算资源（包括软件）已被分配到多个客户端平台上。虽然可以设计出各种不同的分布式环境，但典型的集中式应用已被实施再工程以使之转换为客户 / 服务器（C/S）体系结构，它具有以下特征：应用的功能可以迁移到每个客户端计算机，在客户端可以实现新的 GUI 界面，数据库功能由服务器完成，特殊功能（例如，计算密集型的分析）可以保留在服务器端，必须在客户端和服务器端同时建立新的通信、安全、归档和控制需求。值得注意的是，从集中式计算到 C/S 计算模式的迁移需要同时进行业务再工程和软件再工程。

针对 C/S 应用的再工程一般从对现有大型机的业务环境的彻底分析开始。可以确定三个抽象层次。数据库（database）是客户 / 服务器体系结构的基础，负责管理来自服务器端应用的事务和查询，而这些事务和查询必须控制在一组业务规则（由现有的或再工程后的业务过程所定义）范围内。客户端应用提供面向用户的目标功能。

提问 当面对设计和实现均非常差的程序时，我们有哪些选择？

建议 再工程在很大程度上类似于清洁牙齿。你可以想出上千种理由来拖延它，你可以延迟一会儿，但是最终你的逃离策略会反过来困扰你。

811

建议 在某些情况下，向 C/S 体系结构的迁移不应该作为再工程项目，而应该作为新的开发项目。只有在旧系统的功能被集成到新系统的体系结构中时，才考虑作为再工程项目。

在对数据库层进行重设计之前，必须首先对现有数据库管理系统的功能和现有数据库的数据体系结构进行逆向工程。不管是哪一种情况，都需要对 C/S 数据库进行再工程，以保证：能够以一致的方式处理事务，只能由授权用户完成更新，能够强制执行核心业务规则（例如，在删除某厂商记录前，服务器能够保证不存在与该厂商有关的应付款账号、合同或沟通），能够实现高效查询，以及能够建立完善的归档能力。

业务规则层表示同时驻留在客户端和服务器的软件，该软件执行控制和协调任务，以保证处于客户端应用和数据库之间的事务和查询符合已建立的业务过程。

812

客户端应用层实现特定的最终用户群所需要的业务功能，在很多场合，可以将大型机应用分割为一组更小的、再工程后的桌面应用，桌面应用之间的通信（必要时）由业务规则层控制。

C/S 软件设计与再工程的广泛讨论最好留给专门讨论该主题的书籍，感兴趣的读者可参考 [Van02]、[Cou00] 或 [Orf99]。

36.8.2 面向对象体系结构的正向工程

面向对象软件工程已成为很多软件组织选择的开发范型。但是，使用传统方法开发的现有的应用该怎么办呢？在某些情况下，应该保持这些应用的“现状”，而在另一些情况下，必须对旧的应用实施再工程，使得它们能够容易地集成到大型的、面向对象的系统中。

对传统软件进行再工程以使其成为面向对象的实现，这需要用到本书第二部分讨论的很多技术。首先，要将现有的软件进行逆向工程，以建立适当的数据、功能和行为模型。如果实施再工程的系统扩展了原应用的功能或行为，则还要创建相应的用例（第 8 章和第 9 章）。然后，联合使用在逆向工程中创建的数据模型以及 CRC 建模技术（第 10 章），以奠定类定义的基础。最后，定义类层次、对象关系模型、对象行为模型以及子系统，并开始面向对象的设计。

随着面向对象的正向工程从分析进展到设计，可启用 CBSE 过程模型（第 10 章）。如果旧的应用所在的领域已经存在很多面向对象的应用，则很可能已存在一个健壮的构件库，可以在正向工程中使用它。

对那些必须从头开发的类，有可能复用现有传统应用的算法和数据结构，但是，必须重新设计这些算法和数据结构，以符合面向对象的体系结构。

引述 你可以选择现在付出一点，或者选择日后付出一大笔。
汽车经销商对发动机调整的建议

36.9 再工程经济学

在理想世界中，应该立即淘汰每一个不可维护的程序，而由运用现代软件工程实践开发的高质量的、再工程后的应用所替代。但是，我们生活在一个资源有限的世界，再工程要消耗可能用于其他业务目的的资源，因此，一个组织在试图对现有应用实施再工程之前，应该进行成本效益分析。

813

Sneed[Sne95] 提出了再工程的成本效益分析模型，其中定义了 9 个参数：

P_1 = 应用当前的年度维护成本

P_2 = 应用当前的年度运作成本

P_3 = 应用当前的年度业务价值

P_4 = 再工程后预期的年度维护成本

P_3 = 再工程后预期的年度运作成本

P_6 = 再工程后预期的年度业务价值

P_7 = 估计的再工程成本

P_8 = 估计的再工程日程

P_9 = 再工程风险因子 ($P_9 = 1.0$ 为额定值)

L = 期望的系统生命期

与某候选应用 (即未执行再工程的应用) 的持续维护相关的成本可以定义为:

$$C_{\text{维护}} = (P_3 - (P_1 + P_2)) \times L \quad (36.1)$$

与再工程相关的成本用下面的关系定义:

$$C_{\text{再工程}} = P_6 - (P_4 + P_5) \times (L - P_8) - (P_7 \times P_9) \quad (36.2)$$

利用式 (36.1) 和式 (36.2) 中计算出的成本, 可以计算出再工程的整体效益:

$$\text{成本效益} = C_{\text{再工程}} - C_{\text{维护}} \quad (36.3)$$

可以对所有在库存目录分析 (36.4.2 节) 中标识为高优先级的应用进行上述成本效益分析, 那些显示最高成本效益的应用可以作为再工程对象, 而其他应用的再工程可以推迟到有足够资源时再进行。

36.10 小结

软件维护与软件支持是整个应用生命周期中的持续活动。在软件维护与软件支持活动中, 我们改正了缺陷, 做了适应性修改以适应不断变化的运行环境或业务环境, 在利益相关者的要求下完成了增强任务, 而且能够支持用户将应用集成到他们的个人 workflows 或业务工作中。

再工程发生在两个不同的抽象层次。在业务层, 再工程着重于业务过程, 目的是改变业务过程以提高在某业务领域的竞争力; 在软件层, 再工程考察信息系统和应用, 目的是对它们进行重构以提高质量。

业务过程再工程 (BPR) 能够制定业务目标; 识别并评估现有业务过程 (在确定的目标范围内); 详细描述并设计修订过程; 并在业务中对它们进行原型化、细化和实例化。BPR 的关注点也可以扩展到软件之外, BPR 的结果经常是获得若干方法, 这些方法使信息技术能够更好地支持业务。

软件再工程包括一系列的活动: 库存目录分析、文档重构、逆向工程、程序和数据重构以及正向工程。这些活动的目的是创建现有程序的更高质量和更易于维护的版本——在 21 世纪具有良好生命力的程序。

可以定量地确定再工程的成本效益。将现状的成本 (即与某现有应用不断发生的支持和维护相关的成本) 与预期的再工程成本及维护成本和支持成本的减少进行比较, 对于几乎所有生命期长且当前的可维护性或支持性较差的程序, 再工程代表了成本合算的业务策略。

习题与思考题

- 36.1 考虑你在过去五年中从事过的任何工作, 描述你在其中工作的业务过程。建议使用 36.4.2 节中描述的 BPR 模型修改该过程以提高其效率。
- 36.2 对业务过程再工程的功效做些研究, 给出该方法的正面及反面论据。
- 36.3 老师从班上每个人在本课程中开发的程序中选择一个, 随机地将你的程序和其他人的程序交换,

不对该程序进行解释或走查。现在, 对你所拿到的程序实现某些改进 (由老师指定)。

a. 完成所有软件工程任务, 包括粗略的走查 (但不能和程序的作者交流)。

b. 对测试中遇到的所有错误仔细跟踪。

c. 在班上介绍你的经验。

36.4 探讨在 SEPA 网站上列出的库存目录分析检查表, 然后尝试开发一个适用于现有程序的定量软件评价系统, 目的是从中挑选出再工程的候选程序。你的系统应该扩展到 36.9 节所述的经济分析。

36.5 提出一种纸、墨或传统的电子文档的替代物, 可将它作为文档重构的基础。(提示: 考虑能够用于软件交流的新的描述技术。)

36.6 有人相信人工智能技术将提高逆向工程过程的抽象层次, 对此专题 (也就是说, 人工智能在逆向工程中的应用) 进行研究, 并撰写一篇支持此论点的简短论文。

36.7 为什么当抽象层次增加时, 完备性更难于达到?

36.8 如果完备性增加, 为什么交互性也必须增加?

815 36.9 使用通过 Web 获得的信息, 向班级介绍三种逆向工程工具的特点。

36.10 重构和正向工程之间存在差别, 这种不同是什么?

36.11 对文献和 Internet 资源进行研究, 找出至少一篇讨论从大型机到客户 / 服务器模式的再工程案例研究的文章, 给出概要介绍。

36.12 如何在 36.9 节给出的成本-效益模型中确定 $P_4 \sim P_7$?

扩展阅读与信息资源

具有讽刺意味的是, 软件维护与软件支持是应用生命期中花费成本最高的活动。但是, 描写维护和支持的书要比描写任何其他软件工程主题的书少得多。Reifer (《software maintenance success recipes》, Auerbach, 2011)、Jarzabek (《Effective Software Maintenance and Evolution》, Auerbach, 2007)、Grubb 和 Takang (《Software Maintenance: Concepts and Practice, 2nd edition》, World Scientific Publishing Co., 2003) 以及 Pigoski (《Practical Software Maintenance》, Wiley, 1996) 的书都是最近加入到文献中的。这些书涵盖了基本的维护和支持实践, 而且介绍了实用的管理指南。Schneberger (《Client/Server Software Maintenance》, McGraw-Hill, 1997) 论述了 C/S 环境下的维护技术。Mens 和 Demeyer 编辑的文集 (《Software Evolution》, Springer, 2008) 收集了有关“软件演化”的最新研究。

和很多商业社会的热点话题一样, 围绕业务过程再工程的大肆宣传已经让路给对该主题的更实际的见解。Hammer 和 Champy 的畅销书 (《Reengineering the Corporation, revised edition》, HarperBusiness, 2003) 促进了人们对此的早期兴趣。Jacka 和 Keller (《Business Process Mapping: Improving Customer Satisfaction》, 2nd ed. Wiley, 2009)、Sharp 和 McDermott (《Workflow Modeling》, Artech House, 2nd ed. 2008)、Andersen (《Business Process Improvement Toolbox》, American Society for Quality, 2nd ed. 2007)、Smith 和 Finger [Business Process Management (BPM): the third wave, Meghan-Kiffer Press, 2003] 以及 Harrington 等 (《Business Process Improvement Workbook》, McGraw-Hill, 1997) 所著的书籍给出了 BPR 的案例研究及详细指南。

Abfalter (software Reengineering, VDM Verlag, 2008) 以及 Fong (《Information System Reengineering and Integration》, Springer, 2006) 论述了适用于大部分信息系统的数据库转换技术、逆向工程及正向工程。Nierstrasz 及其同事 (《Object Oriented Reengineering Patterns》, Square Bracket Associates, 2009) 针对面向对象系统的重构和再工程提出了基于模式的观点。Secord 和他的同事 (《Modernizing

Legacy Systems 》, Addison-Wesley, 2003)、Ulrich (《 Legacy Systems : Transformation Strategies 》, Prentice-Hall, 2002)、Valenti (《 Successful Software Reengineering 》, IRM Press, 2002) 以及 Rada (《 Reengineering Software : How to Reuse Programming to Build New, State-of-the-Art Software 》, Fitzroy Dearborn Publishers, 1999) 关注技术层的再工程策略及实践。Miller (《 Reengineering Legacy Software Systems 》, Digital Press, 1998) “提供了保持应用与业务策略及技术改变同步的框架”。

Cameron (《 Reengineering Business for Success in the Internet Age 》, Computer Technology Research, 2000) 和 Umar (《 Application (Re) engineering : Building Web-Based Applications and Dealing with Legacies 》, Prentice-Hall, 1997) 为希望将遗留系统转换为基于 Web 环境的组织提供了有价值的指南。Cook (《 Building Enterprise Information Architectures : Reengineering Information Systems 》, Prentice-Hall, 1996) 讨论了 BPR 和信息技术之间的桥接。Aiken (《 Data Reverse Engineering 》, McGraw-Hill, 1996) 讨论了如何修复、重新组织和复用管理数据。Arnold (《 Software Reengineering 》, IEEE Computer Society Press, 1993) 出版了一本关注软件再工程技术的早期重要论文的优秀文选。

大量关于软件再工程的信息可从网上获得, 最新的与软件维护和再工程相关的参考文献见 SEPA 网站 www.mhhe.com/pressman。

软件工程高级课题

在本书这一部分中，我们考虑一些能够对软件工程加深理解的高级研究课题。在下面的章节中，我们将讨论以下问题：

- 什么是软件过程改进（Software Process Improvement, SPI）？怎样利用软件过程改进来提高软件工程实践的现状？
- 对未来十年软件工程实践有重大影响的发展趋势是什么？
- 软件工程师未来的发展方向是什么？
- 通过对这些问题的回答，将有助于理解在不远的将来对软件工程有深远影响的那些课题。

软件过程改进

要点浏览

概念: 软件过程改进 (SPI) 包含了一系列活动, 这些活动可以产生更好的软件过程, 因而, 更高质量的软件就可以及时地交付给客户。

人员: 主持 SPI 的人员可分为三组: 技术管理者、软件工程师和承担质量保证责任的个人。

重要性: 一些软件组织更关注特别的软件过程。当他们努力改进软件工程实践的时候, 他们不得不关注已有过程中的缺陷, 并且尽量改进他们在软件工作中所采用的方法。

步骤: SPI 方法是迭代的和连续的, 它包

括 5 个步骤: (1) 当前软件过程的评估; (2) 对业务人员和管理者的教育和培训; (3) 过程要素、软件工程方法以及工具的选取和合理性判定; (4) SPI 计划的实现; (5) 基于计划结果的评价和调整。

工作产品: 尽管有很多中间的 SPI 产品, 但最终的结果还是改进软件过程, 并产生更高质量的软件。

质量保证措施: 软件将以更少的缺陷提交给客户, 软件过程中每一阶段的返工将会减少, 及时交付产品的可能性将会得到提高。

很久以前, “软件过程改进” 这个术语就被广泛使用了, 我曾工作过的大多数公司都试图改进他们的软件工程实践的状况。因此我根据经验编写了《Making Software Engineering Happen》这本书 [Pre88]。在这本书的序言中, 有下面这段话:

过去的十年, 我有一些机会帮助很多大型公司实现他们的软件工程实践。这个工作是很困难的, 并且很少能如人们希望的那样顺利——但是, 一旦成功, 其意义就是深远的。软件项目更可能按时完成, 软件开发中涉及的所有人员之间的交流会得到改善。在大型软件项目中的混淆和混乱程度往往会大幅减少。客户遇到的错误数大幅度下降。软件组织的信誉在提高。管理上也少了一个需要担心的问题。

但是, 所有这些并不都是令人愉快的和充满光明的。许多公司试图实施软件工程实践, 但在受挫后不得不放弃。其他一些公司也半途而废, 从来没有看到如上所述的那些好处。还有一些公司以一种严格的方式做了尝试, 其结果是技术人员和管理人员公开抵触, 随后导致士气低落。

尽管这些话是 20 多年前写的, 但今天依然适用。

现在, 绝大多数的软件工程组织都试图 “使软件工程成为现实”。一些组织已经实现了

关键概念

评估
CMMI
教育和培训
评价
设置 / 迁移
合理性判定
成熟度模型
人员 CMM
投资收益率
风险管理
选择
软件过程改进
适应性
框架
过程

一些个别的实践，用来帮助改进其产品的质量，并且提高了交付的及时性。另外一些组织建立了“成熟的”软件过程，用来指导他们的技术和项目管理活动。但是，还有一些组织仍然在努力摸索。他们的实践是碰巧的，过程也是特别的。偶尔，他们的工作也是非常杰出的，但是，更主要的是他们的每项项目都在冒险，没有人知道结局是好还是坏。所以，上面提到的后两种组织都需要软件过程改进吗？答案（可能会令你大吃一惊）是肯定的。那些已经成功地使软件工程成为现实的组织并不能自鸣得意。他们必须继续工作来改进软件工程方法。那些还在努力摸索的组织更要朝着改进的道路前进。

37.1 什么是 SPI

术语软件过程改进（Software Process Improvement, SPI）包含很多方面。首先，它包含以有效方式定义的有效过程的一些要素；其次，组织内已存在的关于软件开发的方法要依据这些要素进行评估；第三，它定义了有价值的改进策略。SPI 策略将已有软件开发的方法转换成一些更集中、更可重复、更可靠的事物（就所生产产品的质量和交付给客户的时间而言）。

关键点 SPI 意味着一个已定义的软件过程、一种组织方法和一种改进策略。

由于 SPI 需要有投入，因此它必然会产生相应的投资回报。实施 SPI 策略所付出的工作量和时间必须要有某种度量方法。这样，改进过程和实践的结果必然会减少解决软件“问题”的费用和时间。必须减少交付给最终用户的软件中存在的缺陷，减少由于质量问题导致的返工次数，减少软件维护和支持（第 36 章）的相关费用，并减少软件延期交付导致的间接成本。

37.1.1 SPI 的方法

尽管一个组织可能会选择不太正式的 SPI 方法，但大多数组织还是会从众多的 SPI 框架中选择一个框架。SPI 框架定义了以下内容：（1）如果要获得有效的软件过程，就要给定一组特性；（2）用来评估是否具有这些特性的一种方法；（3）一种总结这些评估结果的机制；（4）用来帮助软件组织弥补在实施过程中发现弱点和缺失的一种策略。

引述 大多数软件危机是自己造成的，正如一位 CIO 所说：“我宁愿做错也不愿迟做，以后总会有机会修正它。”

Mark Paulk

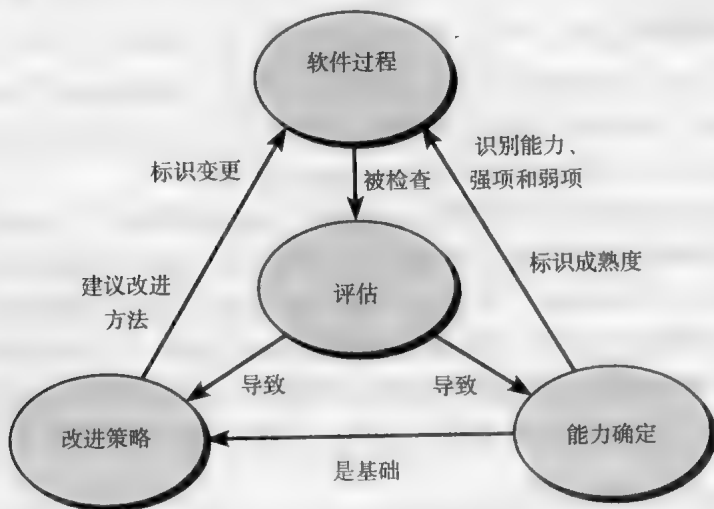


图 37-1 SPI 框架的要素 [Rou02]

SPI 框架评估一个组织软件过程的“成熟度”，并提供成熟度等级定性的表示。实际上，术语“成熟度模型”（37.1.2 节）经常会用到。从本质上讲，SPI 框架包括一个成熟度模型，此模型又包含了一组过程质量指标，这些指标提供了对过程质量的整体测度，而过程质量决定了产品质量。

图 37-1 提供了一个典型的 SPI 框架视图。该框架的关键元素和它们彼此之间的关系如图所示。

应该注意到，不存在通用的 SPI 框架。实际上，一个组织选择的 SPI 框架关系到主持 SPI 工作的相关人员。Conradi[Con96] 定义了 6 种不同的支持 SPI 的相关人员：

提问 谁来支持软件过程改进？

质量认证人员。这类人员支持的过程改进工作关注以下关系：

质量（过程） \Rightarrow 质量（产品）

他们的方法是强调评估方法，并检查一组明确定义的特性，这些特性允许他们确定过程是否展示了质量。他们最可能采用的过程框架有 CMMI、SPICE、TickIT 或 Bootstrap^①。

形式主义者。这部分人想理解（并且如果可能就尽可能地优化）过程 workflow。要完成这些，他们使用过程建模语言（Process Modeling Language, PML）对已存在的过程创建模型，然后进行设计扩展或修改，使过程更加高效。

工具倡导者。这部分人坚持有工具辅助的 SPI 方法，该方法用 workflow 或其他过程特性建模，这种方式可以对改进进行分析。

业务人员。这些人员使用务实的方法，“强调主流项目管理、质量管理和产品管理，应用项目级的计划和度量，但是很少有形式化的过程建模或规则的支持”[Con96]。

改革者。这些人的目标是组织变革，这种变革可能会导致更好的软件过程。他们往往把重点放在人的问题上（37.4 节），更强调人的能力和结构的测度。

理论家。针对特定应用领域或组织结构，这部分人关注特定过程模型的适合性。与典型的软件过程模型（如迭代模型）相比，它们对支持复用或再工程的过程更有兴趣。

随着 SPI 框架的应用，发起 SPI 的相关人员（不管其总体重点如何）必须建立一些机制以达到以下目的：（1）支持技术变迁；（2）确定一个组织吸收所提出的过程变革的准备程度；（3）衡量已经采取变革的程度。

37.1.2 成熟度模型

成熟度模型在 SPI 框架环境中应用。成熟度模型的目的是提供软件组织所具有的“过程成熟度”的总体指标，即软件过程质量的指标、业务人员对过程理解和应用的程度、软件工程实践的总体状况。这可以使用一些有顺序的等级来达到。

例如，卡内基·梅隆大学软件工程研究所的能力成熟度模型（37.3 节）给出了 5 个成熟度等级，从初始级（最基本的软件过程）到优化级（取得最佳实践的过程^②）。但经验表明，许多组织展示的“过程不成熟”的等级破坏了改进软件工程实践的合理尝试。Schorsch [Sch96] 建议了 4 个不成熟级别，这些都常常在现实世界的软件开发组织中遇到，这 4 个不成熟级别如下。

关键点 成熟度模型定义了软件过程能力和执行的水平。

① 这些 SPI 框架会在本章后面讨论。

② 37.3 节讨论了 CMM 模型的更新。

第 0 级，粗心大意。无法让成功的开发过程取得成功。认为所有的问题都是技术问题。管理和质量保证活动被认为是软件开发过程中多余的开销。依靠银弹。

第 1 级，故意阻碍。强加了一些适得其反的过程。过程是严格定义的并且强调对形式的固守。华而不实的仪式比比皆是。集体管理阻碍责任的分配。所有的事情都维持现状。

第 2 级，蔑视。漠视良好的软件工程制度化。完全分裂软件开发活动和软件过程改进活动。缺乏完整的培训计划。

第 3 级，暗中破坏。完全忽视自己的章程，蓄意诋毁同行组织软件过程改进的努力。鼓励失败和不良行为。

Schorsch 的不成熟级别对软件组织是有危害的。如果你遇到它们中的某一个，SPI 尝试注定要失败。

最主要的问题是成熟度级别（例如作为 CMM 一部分提出来的）是否带来了实质的好处。我认为的是。成熟度级别提供了易于理解的过程质量快照，业务人员和管理者可以将其作为参考基准使用，从中规划改进的策略。

37.1.3 SPI 适合每个人吗

很多年来，SPI 一直被视为“大型企业”的活动——仅仅在大公司起作用的一种委婉说法。但是今天，职员少于 100 人的所有软件开发公司中，有相当大比例公司都用到了 SPI。一个小型软件公司能真正用到 SPI 活动并保证它成功吗？

在大型软件开发组织和小型软件开发组织之间存在重大的文化差异。小型组织更加非正式，很少应用标准实践，倾向于自我管理，这不足为奇。他们对软件组织个别成员的“创造力”还往往感到自豪，并且最初以为 SPI 框架过于官僚和笨重。然而，过程改进对于小型软件开发组织和大型软件开发组织一样，都是非常重要的。

在小型组织内，实施 SPI 框架所需要的资源可能是组织内缺少的。管理者必须分配相应的人力和财力，以使软件工程成为现实。因此，不管软件组织的规模是大是小，考虑实施 SPI 的商业动机都是合理的。

只有当 SPI 的支持者证明了它的财务杠杆作用，SPI 才会被核准并实施 [Bir98]。财务杠杆通过检查技术益处（例如，交付时更少的缺陷、减少返工、更低的维护成本或更快的上市时间）并将它们转化成金钱来证明。本质上，你必须对 SPI 成本提供现实的投资回报（37.7 节）。

建议 如果一个特定的过程模型或 SPI 方法过度地伤害了你的组织，那也没有什么稀奇，很可能就是这样。

822

37.2 SPI 过程

使用 SPI 的困难之处不是定义一组特性来描述高质量的软件过程或过程成熟度模型的创建。这些东西是比较容易的。相反，最困难的是就如何在整个软件组织中对启动 SPI 以及制定 SPI 不断前进的策略达成共识。

卡内基·梅隆大学软件工程研究所已经开发了 IDEAL——“一个组织改进模型，作为路线图服务于启动、策划和实施改进活动” [SEI08]。IDEAL 是很多 SPI 过程模型的代表，它定义了 5 个不同的活动——启动、诊断、建立、行动和学习，通过这些 SPI 活动来指导组织进行实践。

在本书中，基于原来在 [Pre88] 中提出的 SPI 过程模型，我们提出了一个有些不同的

SPI 路线图。该模型应用常识哲学，要求组织做到：（1）自我检查；（2）使过程更敏捷以便于做出智慧的选择；（3）选择能最好地满足其需求的过程模型（以及相关的技术要素）；（4）在组织的运行环境和组织文化内将该模型实例化；（5）评价完成的工作。这 5 项活动（随后讨论^①）以一种迭代（循环）的方式应用，以便于促进持续的过程改进。

37.2.1 评估和差距分析

若先不评估当前框架活动和相关的软件工程实践的有效性，而是试图改善当前的软件过程，任何这样的尝试都像是到一个新地方的漫长旅途的开始，而你却又不知道从哪里开始。你会充满兴致地出发，四处徘徊，试图弄清自己的处境，花费了大量的精力，承受挫折带来的痛苦。很可能，你决定真的不想再走了。简单地说，在你开始任何旅行之前，最好先准确地知道你在哪里。

路线图上的第一项活动是评估，即让你弄清自己的处境。评估的目的是揭示组织以某种方式应用现有软件过程及构成此过程的软件工程实践的优势和劣势。

建议 一定要真正弄清你的优势和劣势，并且明智地把工作建立在优势上。

评估在很大的范围内检查活动和任务，这将导致高质量的过程。例如，不管选择什么样的过程模型，软件组织必须建立通用的机制，如：定义与客户沟通的方法；建立表示用户需求的方法；定义包括范围、估计、进度要求以及项目跟踪的项目管理框架；风险分析方法；变更管理规程；质量保证以及包括评审的控制活动等。在已经建立的框架活动和普适性活动（第 3 章）中，每项活动都经过了深思熟虑，并且要进行评估，以确定以下问题是否得到了解决：

- 是否清楚地定义了每项活动的目标？
- 是否标识和描述了需要作为输入的工作产品及作为输出产生的工作产品？
- 是否清晰地描述了要执行的工作任务？
- 是否通过角色标识了执行这些活动的人员？
- 入口和出口标准已经建立了吗？
- 活动的度量是否已经建立？
- 支持这些活动的工具是否是可用的？
- 针对这些活动有明确的培训大纲吗？
- 对所有的项目，活动是否一致地执行？

尽管上述问题的答案不是是就是否，但评估过程需要洞察答案背后的原因，以确定问题相关的执行方式是否符合最佳实践。

随着过程评估的实施，你（或其他执行评估的人）应关注以下几个问题。

一致。所有的软件团队是否在所有的软件项目中一致地应用了重要的活动、行动和任务？

成熟。执行了一定成熟级别的管理和技术行动是否意味着对最佳实践有了透彻的理解？

认可。软件过程和软件工程实践是否得到了管理部门和技术人员的广泛认可？

承诺。管理者已经承诺为达到一致、成熟和认可所需要的资源了吗？

^① 经过允许，[Pre88]中的一些内容已经重新改写。

实际情况和最佳实践之间的差异表示存在改进机会的“空间”。在何种程度上能够获得一致、成熟、认可和承诺表明了需要做多少文化上的变更才能获得意义深远的改进。

37.2.2 教育和培训

尽管很少有人怀疑敏捷的、有组织的软件过程或者完整的软件工程实践的好处，但很多从业人员和管理者并非充分了解这些课题^①。因此，在引入 SPI 框架时，对过程和实践的错误认识会导致不恰当的决定。由此得出结论，任何 SPI 策略的关键要素是对从业人员、技术管理人员和直接接触软件组织的高级管理人员的教育和培训。应该进行三种类型的教育和培训：一般的概念和方法；特定的技术和工具；沟通交流和与质量相关的课题。

建议 针对软件团队的真正需要，尽量提供“及时”的培训。

在现代化条件下，教育和培训可以包括各种不同的方式。从播客到简短的 YouTube 视频再到复杂的以互联网络培训（例如 [QAI08]），从 DVD 到教室里的面对面课程，这些都可以作为 SPI 策略的一部分。

37.2.3 选择和合理性判定

一旦完成了最初的评估活动^②，并且教育已经开始，软件组织就应该开始做出选择。这些选择在选择和合理性判定活动期间做出，其中，选择过程特性及特定的软件工程方法和工具在软件过程中占有重要位置。

首先，应该选择最适合你的组织、利益相关者和所开发软件的过程模型（第 3～5 章）。应该决定应用一组框架活动中的哪一个、要生产的主要工作产品以及使团队能够评估进展的质量保证检查点。如果 SPI 评估活动表明了一些特定的弱项（例如，不正规的 SQA 功能），则应该关注那些与弱项直接相关的过程特性。

建议 当你在做选择时，一定要考虑你的组织文化以及对每个选择的接受程度。

其次，对每个框架活动（例如建模）进行工作分解，定义应用于典型项目中的任务集。还应该考虑能完成这些任务的软件工程方法。一旦选定，就应该协调教育和培训，以加强理解。

[825]

理想的情况是每个人都参与选择各种过程和技术要素的工作，并且向着设置和迁移活动（37.2.4 节）顺利过渡。实际上，选择活动可能是一项艰难的活动。在不同的支持者之间达成共识经常是很难的。如果委员会确立了选择的标准，人们可能会喋喋不休地争论标准是否是适当的以及做出的选择是否真正符合已确立的标准。

诚然，不好的选择会比好的选择造成更大的危害。但是“分析麻痹”（指注意力集中在一点上而导致动作不连贯）意味着几乎没有取得任何进展，过程中出现的问题依然存在。只要过程特性或技术要素有满足组织需要的很好时机，有时候，行动起来并且做出选择更好，而不是等待最完美的解决方案。

一旦做出了选择，时间和金钱就必须花在组织内的实例化中，这些资源支出应该是合理的。SPI 成本合理性判定以及投资收益率将在 37.7 节进行讨论。

① 如果你花了很多时间读本书，你就不会像他们一样。

② 实际上，评估是一项持续的活动。它需要定期进行，以确定 SPI 策略是否实现了其近期目标，并设定今后改进的阶段任务。

37.2.4 设置 / 迁移

设置是实施了 SPI 路线图后软件组织感受到的第一项变更效果。在某些情况下, 将一个全新的过程推荐给组织, 必须定义框架活动、软件工程行动以及人员的工作任务, 并作为新的软件工程文化的一部分进行设置。这样的变化表示重要组织和技术的变迁, 需要精心地管理。

在其他情况下, 与 SPI 相关联的变化相对较少, 但对已有的过程模型做了有意义的修改。通常将这样的变化称为过程迁移。今天, 很多软件组织都有恰当的“过程”。问题是, 现有过程的工作效率可能不高。因此, 从一个过程 (不像期望的那样工作) 到另一个过程的增量式迁移是更有效的策略。

设置和迁移实际上是软件过程再设计 (Software Process Redesign, SPR) 活动。Scacchi[Sca00] 认为“SPR 是与识别、应用和细化相关的一种新方式, 并能极大地提高和改变软件过程”。当对 SPR 启动形式化方法时, 需要考虑三个不同的过程模型: (1) 已存在 (“现有”) 过程; (2) 过渡 (“这里到那里”) 过程; (3) 目标 (“将要成为的”) 过程。如果目标过程和已存在过程有很大的差别, 那么设置的唯一合理方法是采用增量策略, 分步执行过渡过程。过渡过程提供了一系列导航点, 能保证软件组织文化经过一段时间后可以适应一些小的变化。

37.2.5 评价

尽管我们将评价列为 SPI 路线图的最后一项活动, 但其在整个 SPI 中都存在。评价活动评估设置及采纳变更的程度、这些变更在多大程度上提高了软件质量或其他可见的过程收益, 以及随着 SPI 活动的进行过程和企业文化的总体状况如何。

在评价活动中, 定性因素和定量度量都要考虑。从定性的角度看, 过去的管理者和业务人员对软件过程的态度可以和设置过程后接受调查的态度进行对比。定量度量 (第 32 章) 是从已经使用过渡过程或目标过程的项目中收集的, 并与从使用现有过程的项目中所收集的类似度量进行比较。

37.2.6 SPI 的风险管理

SPI 是有风险的。实际上, 在所有 SPI 尝试中, 一半以上都以失败而告终。失败的原因各不相同, 且与特定的组织有关。最普遍的风险是: 缺少管理者的支持, 技术人员文化上的抗拒, 规划糟糕的 SPI 策略, SPI 过度形式化的方法, 选择了不恰当的过程, 缺少主要利益相关者的投资, 不合适的预算, 工作人员缺少培训, 组织的不稳定以及很多其他因素。这些因素均可用于分析可能的风险, 并制定减轻这些风险的内部策略。

关键点 SPI 经常失败, 其原因没有恰当地考虑风险, 也没有制定应急计划。

软件组织应该就 SPI 过程从以下三个关键点进行风险管理 [Sta97b]: 在启动 SPI 路线图之前, 在执行 SPI 活动 (评估、教育、选择、设置) 期间, 以及一些过程特性实例化之后的评估活动期间。一般可以对 SPI 风险因素进行下面的分类 [Sta97b]: 预算和成本、内容和交付、文化、SPI 交付物的维护、任务和目标、组织的管理者、组织的稳定性、过程的利益相关者、SPI 工作开展的时间安排、SPI 工作开展环境、SPI 工作开展过程、SPI 项目管理以及 SPI 工作人员。

在每一类中都定义了很多通用的风险因素。例如，组织文化对风险产生重大的影响。从组织文化方面，可以定义如下的通用风险因素[⊖][Sta97b]：

- 对变革的态度，这与对变革的前期工作量投入相关。
- 与质量大纲相关的经验，成功的程度。
- 解决问题的行动方向与对方针的争论。
- 使用事实管理组织和业务。
- 对改变的耐心，花时间参与交往的能力。
- 提倡采用工具——期望工具能够解决问题。
- “计划充实性”的等级——对计划的组织能力。
- 组织成员在各级组织会议上公开的参与能力。
- 组织成员有效地掌控会议的能力。
- 在有明确定义的过程组织中经验的等级。

827

使用风险因素和通用属性作为指导，我们开发了一个风险表（第 35 章）以保证管理者的进一步关注。

37.3 CMMI

作为完整的 SPI 框架，最初的 CMM 是由卡内基·梅隆大学软件工程研究所（Software Engineering Institute, SEI）在 20 世纪 90 年代开发并升级的。今天，它已经演变为能力成熟度模型集成（Capability Maturity Model Integration, CMMI）[CMM07]，它是一个综合的过程元模型，以一组系统工程和软件工程能力为基础，能够表示组织可以达到的过程能力及成熟度的不同等级。

CMMI 以两种不同的方式表示过程元模型：（1）作为一个“连续式”模型；（2）作为一个“分级式”模型。“连续式”CMMI 元模型以两个维度描述过程，如图 37-2 所示。每个过程域（例如，项目计划或需求管理）根据特定目标和特定实践进行正式评估，并且与下面的能力等级相关联。

能力等级 0：不完全级（incomplete）。过程域（如需求管理）或者没有执行，或者已经执行，但没有达到该过程域 CMMI 1 级成熟度所规定的所有目标。

能力等级 1：已执行级（performed）。（由 CMMI 所定义的）过程域的所有特定目标都已经满足。为生产已规定工作产品所需的工作任务都已执行。

能力等级 2：已管理级（managed）。能力等级 1 中所有的标准都已经满足。此外，所有与过程域相关的工作都符合组织规定的方针，所有的工作人员都可以得到完成工作所需的足够资源，利益相关者都按照需要积极地投入到过程域中，所有的工作任务和工作产品都可以被“监督、控制和评审，并评估是否与过程描述相一致”[CMM07]。

能力等级 3：已定义级（defined）。能力等级 2 中所有的标准都已经满足。

另外，这个过程是“根据组织剪裁准则，对其标准过程进行了剪裁，剪裁过的过程对组织的过程资产增添了新的内容，如工作产品、测量和其他过程改进信息

建议 每个组织都应该努力获得 CMMI 的真正意图。然而，模型每个方面的实现都有可能对你的现状具有矫枉过正的影响。

828

⊖ 本节提到的每类风险因素都可以在 [Sta97b] 中找到。

等” [CMM07]。

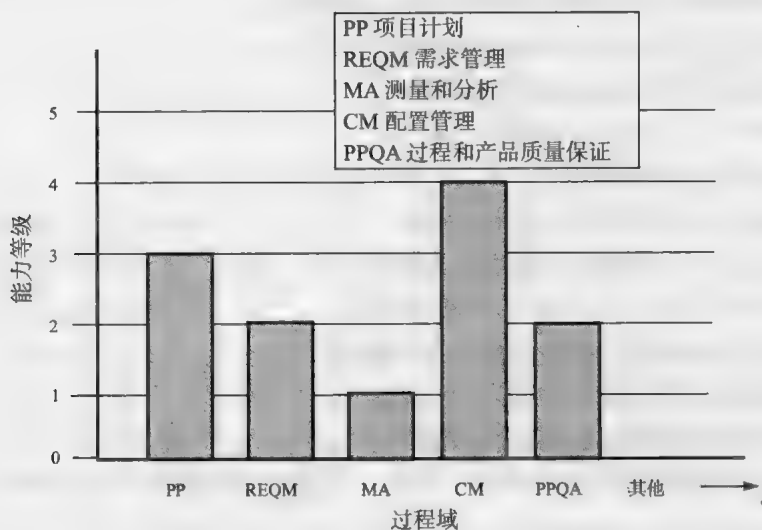


图 37-2 CMMI 过程域能力分布图 [Phi02]

能力等级 4：定量管理级 (quantitatively managed)。能力等级 3 中所有的标准都已经满足。此外，通过采用测量和定量评估等手段，对过程域进行控制和不断改进。“已经建立起来对质量和过程性能的定量指标，并作为过程管理的标准”。[CMM07]

能力等级 5：优化级 (optimized)。能力等级 4 中所有的标准都已经满足。此外，采用定量 (统计) 的方法调整和优化过程域，以满足用户不断变更的需求，并持续地提高过程域的有效性。

CMMI 定义了每个过程域的“特定目标”，以及为达到这些目标所需的“特定实践”。特定目标明确了如果该过程域的所有活动都有效执行的话，软件过程应具备的特点。特定实践将目标细化成一组过程相关的活动。

例如，项目计划是 CMMI 为“项目管理”类定义的 8 个过程域之[⊖]。为项目计划定义的特定目标 (Specific Goal, SG) 和相关的特定实践 (Specific Practice, SP) 如下 [CMM07]：

SG 1 确立估计值

SP 1.1-1 确立项目范围

SP 1.2-1 建立工作产品和任务属性的估计值

SP 1.3-1 定义项目的生命周期

SP 1.4-1 确定工作量和成本估计值

SG 2 制定项目计划

SP 2.1-1 编制预算和进度计划表

SP 2.2-1 识别项目风险

SP 2.3-1 制定数据管理计划

SP 2.4-1 制定项目资源计划

网络资源 关于 CMMI 的完整信息和可下载版本可以从 <http://cmmi-institute.com/resources/> 获得。

[⊖] 为“项目管理”定义的其他过程域包括：项目监控、供应商协议管理、IPPD 的集成项目管理、风险管理、集成团队建立、集成供应商管理以及定量项目管理。

- SP 2.5-1 制定所需知识技能计划
- SP 2.6-1 制定利益相关者的参与计划
- SP 2.7-1 制定项目计划
- SG 3 获取对计划的承诺
 - SP 3.1-1 评审影响项目的计划
 - SP 3.2-1 使工作和资源投入相匹配
 - SP 3.3-1 获得对计划的承诺

除了特定目标和特定实践,CMMI 还为每个过程域定义了一组 5 个通用目标和相关实践。其中,每个通用目标都对应着 5 个能力等级之一。因此,要想达到特定的能力等级,就必须满足该等级对应的通用目标和相应的通用实践。

阶段性的 CMMI 过程模型定义了和持续性模型相同的过程域、目标和实践。主要区别是阶段性模型定义了 5 个成熟度等级,而不是 5 个能力等级。要达到某个成熟度等级,就必须实现与一组过程域相关的特定目标和特定实践。成熟度等级和过程域之间的关系如图 37-3 所示。

830

等级	焦点	过程域
优化级	持续过程改进	组织创新和部署 原因分析和消除
定量管理级	定量管理	组织过程绩效 定量项目管理
已定义级	过程标准化	需求开发 技术解决方案 产品集成 验证 确认 组织过程焦点 组织过程定义 组织培训 集成项目管理 集成供应商管理 风险管理 决策分析和决定 组织集成环境 集成团队建立
已管理级	基本的项目管理	需求管理 项目计划 项目监控 供应商协议管理 测量和分析 过程和产品质量保证 配置管理
初始级		

图 37-3 达到成熟度等级所需的过程域 [Phi02]

信息栏 CMMI——应不应该使用?

CMMI 是一个过程元模型。它(用 700 多页)定义了软件组织想建立完整的软件过程应该具备的过程特性。已经争论了十多年的问题是:“CMMI 是否执行过度了?”像日常生活中(以及软件中)的大多数事情一样,答案不是简单的“是”或“否”。

SPI 的精神总是应该被接纳。为避免过于简化的风险,CMMI 认为软件开发过程必须严肃对待——必须计划周全,必须统一控制,必须准确跟踪以及必须专业化地执行。必须关注项目利益相关者的需要、软件工程师的技能以及最终产品的质量。任何人都应该质疑上述观点。

如果软件组织要构建大型复杂的系统,此系统需要几十人或几百人参与、

历时几个月或很多年,就应该认真考虑 CMMI 的具体要求。如果组织的文化符合标准过程模型,并且管理者又承诺要使其获得成功,这也许正是适合使用 CMMI 的场合。然而,在其他情况下,CMMI 的内容可能是太多了,组织不能很好地采纳。这意味着 CMMI “很差”或“过于官僚”或“老式”吗?不,绝不是这样。它只是意味着适用于一种组织文化的东西可能并不适用于另一种组织文化。

CMMI 是软件工程的一项重要成就。对于在构建计算机软件时到底采用哪些活动和动作,CMMI 提供了全面的讨论。即使软件组织不采用它的细节,也都应该接受它的精神,并且从 CMMI 对软件过程和实践的讨论中得到启发。

37.4 人员 CMM

不管设想得多好,如果没有具备才能和积极性的软件人才,软件过程也不会成功。人员能力成熟度模型(People Capability Maturity Model)“是实现劳动力实践的路线图,这些实践可以持续提高软件组织中员工的能力”[Cur01]。人员 CMM 的目标是 20 世纪 90 年代中期开发的,其目标在于鼓励普通员工在知识(称为“核心能力”)、具体软件工程技能和项目管理技能(称为“劳动力能力”)以及过程相关能力诸方面的持续提高。

关键点 人员 CMM 建议通过实践提高劳动力的技能和文化。

像 CMM、CMMI 以及相关的 SPI 框架一样,人员 CMM 定义了组织成熟度的 5 个等级,提供了关于员工实践和过程的相对成熟的指标。这些成熟度等级[CMM08]与已有的一组关键过程域(Key Process Area, KPA)(在一个组织内)相连。组织等级和相关的 KPA 概况如图 37-4 所示。人员 CMM 通过鼓励组织培养和改进其最重要的资产(即人员)来补充 SPI 框架。同样重要的是,它在员工中形成一种氛围,使软件组织能够“吸引、培养及留住优秀人才”[CMM08]。

37.5 其他 SPI 框架

尽管卡内基·梅隆大学软件工程研究所的 CMM 和 CMMI 是应用最广泛的 SPI 框架,但仍然有一些其他框架^①被提出及应用。以下对这些 SPI 框架作简要介绍^②。

① 合理地说,这些框架中的一些并不是“备选框架”,因为它们只是 SPI 方法的补充。更多有关 SPI 框架的综合列表可以在 <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.4787&rep=rep1&type=pdf> 找到。

② 如果你有更多的兴趣,可以查阅其他印刷品或网络资料。

等级	焦点	过程域
优化级	持续改进	持续的员工创新 组织绩效协调 持续的能力提高
可预测级	对知识、技巧和能力进行量化和管理	师徒制 组织能力管理 定量绩效管理 基于能力的估值 授权工作组 能力整合
已定义级	识别和开发知识、技巧和能力	分享文化 工作组开发 基于能力的实践 职业发展 能力发展 员工规划 能力分析
已管理级	可重复的、基本的人员管理实践	薪酬 培训和开发 绩效管理 工作环境 交流和协作 人员配备
初始级	不一致的实践	

图 37-4 人员 CMM 的过程域

SPICE。SPICE（Software Process Improvement and Capability dEtermination）模型提供了遵循 ISO 15504:2003 和 ISO 12207 的 SPI 评估框架。SPICE 文档套件 [SDS08] 描述了一个完整的 SPI 框架，包括过程管理模型，对考虑中的过程进行评估和评级的指南，对评估设备和工具的构造、选择和使用，以及评估师的培训。

Bootstrap。Bootstrap SPI 框架“已经被开发出来，目的是确保与最新的软件过程评估和改进（SPICE）的 ISO 标准相一致，并且与 ISO 12207 相协调”[Boo06]。Bootstrap 的目标是使用一系列最好的软件工程实践作为评估的基础，评价一个软件过程。像 CMMI 一样，Bootstrap 根据收集到的软件过程和软件项目“现状”的调查问卷情况，给出过程成熟度等级。SPI 的指南是基于成熟度等级和组织目标的。

PSP 和 TSP。虽然一般将 SPI 描述为组织活动，但没有理由说明不能在个人或团队级别上进行过程改进。PSP 和 TSP（第 4 章）都强调需要持续地收集进行中的工作数据，并且利用这些数据开发改进策略。PSP 和 TSP 方法的提出者 Watts Humphrey（[Hum97]，[Hum00]）给出了如下评论：

PSP（和 TSP）展示了如何计划和跟踪工作，如何始终如一地生产高质量的软件。使用 PSP（和 TSP）会提供数据来表明工作效率并标识你的强项和弱项。要想获得成功和高薪的职务，就需要知道自己具备的技术和能力，并努力提高，在所从事的工作中利用好自己的独特才能。

提问 除了 CMM 以外，还有别的可供考虑的 SPI 框架吗？

引述 软件组织已经表现出他们从已完成的项目中获取经验的能力明显不足。

NASA

TickIT。TickIT 对方法进行审核 [Tico05] 以确保其与软件 ISO 9001:2000 相一致。ISO 9001:2000 是通用标准，它适用于任何想改进其产品、系统或者服务质量的组织。因此，这个标准可直接应用于软件组织和公司。

ISO 9001:2000 提出的基本策略如下 [ISO01]：

ISO 9001:2000 强调对于一个组织而言识别、实施、管理和持续提高过程有效性的重要性，这对于质量管理体系是必需的，为了达到组织的目标还要管理这些过程之间的相互作用。过程有效性和效率可以通过内部和外部的评审过程来评估，并根据成熟度等级来评价。

ISO 9001:2008 已经采用了“计划-实施-检查-行动”循环，它适用于软件项目的质量管理要素。在软件环境中，为了获得高质量软件并达到客户满意，“计划”要建立所需要的过程目标、活动和任务。“实施”执行软件过程（包括框架活动和普适性活动）。“检查”监控和检测过程，以保证可以达到质量管理所提出的所有要求。“行动”着手于软件过程改进活动，使改进过程持续地进行。可以在整个“计划-实施-检查-行动”周期中使用 TickIT，以保证 SPI 的进展。作为 ISO 9001:2008 认证的先驱，TickIT 审核员评估上述循环的应用情况。关于 ISO 9001:2008 和 TickIT 更详细的讨论可以参考 [Ant06]、[Tri05] 或 [Sch03]。

网络资源 关于 ISO 001: 2008 的优秀总结可以在 <http://praxiom.com/iso-9001.htm> 找到。

37.6 SPI 的投资收益率

SPI 是艰苦的工作，需要投入大量的财力和人力。那些批准 SPI 预算和资源的管理者总是会问这样的问题：“怎样才能知道我们所投入的资金会取得合理的回报？”

在定性的层次上，SPI 的拥护者认为，改进软件过程将会带来软件质量的提高。他们主张，改进了的过程将导致实施更好的质量筛选（结果是减少缺陷的传播）、更好地变更控制（结果是减少项目造成的混乱）、更少的技术返工（从而降低成本并且能获得更好的上市时机）。但是这些定性的收益能转变成定量的结果吗？经典的投资收益率（Return On Investment, ROI）等式如下：

$$ROI = \frac{\sum(\text{收益}) - \sum(\text{成本})}{\sum(\text{成本})} \times 100\%$$

收益包括与更高的产品质量（更少的缺陷）、更少的返工、变更方面减少的工作量相关的成本节省，以及从缩短上市时间中获得的收入。

成本包括直接的 SPI 成本（例如，培训费、测量费），也包括与强调质量控制和变更管理活动以及应用更严格的软件工程方法相关的间接成本（例如，设计模型的创建）。

在现实世界中，这些定量的收益和成本有时是难以准确计量的，并且所有都是可以开放解释的。但这并不意味着软件组织应该对增加的成本和收益不经过仔细的分析就实施 SPI 过程。关于 SPI 的投资收益率更全面的叙述可以在 David Rico 唯一的一本书 [Ric04] 中找到。

37.7 SPI 趋势

在过去的 25 年中，很多公司都在尝试应用 SPI 框架改进他们的软件工程实践，这些 SPI 框架影响了组织的变化和技术的变迁。正如本章前面提到的，超过一半的尝试失败了。不管成功还是失败，都耗费了大量的金钱。David Rico [Ric04] 报道了 SPI 框架的一个典型应用，如 SEICMM 的花费是每人 25000 美元到 70000 美元之间，并且需要数年才能完成。其

实,这毫不奇怪,SPI 的未来应该强调一种成本较低并且费时较少的方法。

为了使 21 世纪的开发更有效率,未来的 SPI 框架必须变得更加敏捷。它不再是以组织为关注点(这可能需要很多年才能够成功完成),而是集中在项目层面上,努力在几个星期内改进团队过程,而不是几个月或几年。要想在很短的时间内取得有意义的成果(即使在项目级别上),复杂的框架模型可能要让位于简单的模型。要求做到十几个关键实践和数以百计的补充实践,不如强调仅有的几个核心实践(例如,类似于本书所讨论的框架活动)。

835

SPI 方面的任何尝试都需要具有一定知识的人员,但是教育和培训费用可能是昂贵的,应该将其最小化(使其流线化)。未来的 SPI 实施工作应该依靠基于网络的培训,定位于核心的实践,而不是课堂上的课程(昂贵和费时)。不要去做改变组织文化(这可能带来涉及组织方针的风险)的深远尝试,正如现实世界中一样,在某一时刻在一个小团体内,直到出现了一个转折点,文化变革才会发生。

在过去的 20 年中,SPI 工作具有重要的价值。已经开发的框架和模型代表了软件工程界重要的智力资产。但是像所有的事情一样,这些资产对 SPI 未来尝试的指导并不是可重复性的教条,而是作为更好的、更简单的和更敏捷的 SPI 模型的基础。

37.8 小结

软件过程改进框架定义了一些特性(要获得有效的软件过程,就必须具备这些特性)、一种评估方法(有助于确定是否具备了这些特性)以及一种策略(辅助软件工程组织实现那些薄弱或缺失的过程特性)。无论发起 SPI 的人是谁,其目标是提高过程质量,进而提高软件的质量和交付的及时性。

过程成熟度模型从总体上体现了软件组织呈现的“过程成熟度”,对于正在使用的软件过程的相对有效性提供了定性的认识。

SPI 路线图开始于评估——一系列的评价活动,通过组织应用的现有软件过程和作用于这些过程的软件工程实践的方式,既揭示了优势,也揭露了缺陷。作为评估的结果,软件组织可以制定一个整体的 SPI 计划。

任何 SPI 计划的关键要素之一是教育和培训,该活动主要关注提高管理者和从业人员的知识水平。一旦工作人员精通了目前的软件技术,选择和合理性判定就可以开始了。这些任务导致对软件过程体系架构、采用的方法和支持工具的选择。设置和评价是 SPI 活动,这些活动将过程的改变具体化,并可以评估其有效性和影响。

要想成功地改进软件过程,一个组织必须具备以下特征:管理者对 SPI 的承诺和支持,工作人员参与 SPI 的整个过程,过程集成到整个组织文化,制定适应本单位要求的 SPI 策略以及 SPI 项目的可靠管理。

836

今天,很多 SPI 框架已经用于软件工程实践。卡内基·梅隆大学软件工程研究所的 CMM 和 CMMI 已经广泛应用。对人员 CMM 进行定制可用以评估组织文化质量和其中的人员。SPICE、Bootstrap、PSP、TSP 和 TickIT 是另外一些有效的 SPI 框架。

SPI 是艰苦的工作,要求投入大量的财力和人力。为了保证获得合理的投资回报,一个组织必须估计与 SPI 相关的费用和直接利益。

习题与思考题

- 37.1 为什么软件组织在开始努力改善其软件过程时经常陷入争端?
- 37.2 用自己的语言描述“过程成熟度”这个概念。
- 37.3 做一些研究(查看 SEI 站点),确定美国和全世界的软件组织的过程成熟度分布情况。
- 37.4 你是一个非常小的软件组织工作——只有 11 个软件开发人员。SPI 适合你吗?解释你的答案。
- 37.5 评估类似于每年的体检。用体检作为比喻,描述一下 SPI 评估活动。
- 37.6 “当前”过程、“过渡”过程和“目标”过程三者之间的区别是什么?
- 37.7 在 SPI 环境中如何实现风险管理?
- 37.8 选择 37.2.7 节中的一个关键成功因素,做一些研究,并撰写一篇简短的论文说明它是如何实现的。
- 37.9 做一些研究,并解释 CMMI 与其前身 CMM 在哪些方面存在差异。
- 37.10 从 37.5 节讨论的 SPI 框架中选择一个,并撰写一篇简短的论文给出更详细的描述。

扩展阅读与信息资源

SPI 方面最容易获取的综合信息资源之一是由卡内基·梅隆大学软件工程研究所(SEI)开发的,其网址是 www.sei.cmu.edu 以及 www.commiinstitute.com。SEI 的网站包含数百篇论文、研究报告和详细的 SPI 框架描述。

在以往 20 年已有的广泛的图书和文献的基础上,过去几年又增加了一些有价值的书籍。Chrissis 和她的同事(《CMMI for Development: Guidelines for Process Integration and Product Improvement》, 3rd ed, Addison-Wesley, 2011)以及 McMahan(《Inegrating CMMI and Agile Development》, Addison-Wesley, 2010)讨论了当今软件开发中 CMMI 的应用。Land(《Jumpstart CMM/CMMI Software Process Improvements》, Wiley-IEEE Computer Society, 2007)将 SEICMM 和 CMMI 的一部分需求与 IEEE 软件工程标准相融合,强调软件过程和实践的交叉。Micklewright(Lean ISO 9001, ASQ Quality Press, 2010)以及 Cianfrani 和他的同事(ISO 9001: 2008 Explained, 3rd ed., ASQ Quality Press, 2009)阐述了 ISO 9001:2008 的内容和含义。Mutafelija 和 Stromberg(《Systematic Process Improvement Using ISO 9001:2000 and CMMI》, Artech House Publishers, 2007)讨论了 ISO 9001:2000 和 CMMI SPI 框架及它们之间的“协同作用”。Conradi 和他的同事(《Software Process Improvement: Results and Experience from the Field》, Springer, 2006)介绍了一系列案例研究和相关的 SPI 实验结果。

Mckay 和 Black(《Improving the Software Process》, RBCS, 2012)对 SPI 做全面的介绍。Fauzi 和他的同事(《Software Process Improvement: Approaches and Tools for Practical Development》, IGI Global, 2011)以及 Van Loon(《Process Assessment and Improvement: A Practical Guild for Managers, Quality Professionals and Assessors》, Springer, 2006)在 ISO/IEC 15504 环境下讨论了 SPI。Watts Humphrey(《PSP》, Addison-Wesley, 2005 和《TSP》, Addison-Wesley, 2005)在两本不同的书籍中描述了个人团队过程 SPI 框架和团队软件过程 SPI 框架。Fantina(《Practical Software Process Improvement》, Artech House Publishers, 2004)以 CMMI/CMM 为重点提供了务实的操作方法指导。

更广泛的关于软件过程改进的信息可以通过网络获取。最新的 SPI 相关参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

软件工程的新趋势

要点浏览

概念: 没有人能够绝对准确地预测未来。但是推测一下软件工程领域未来的趋势并从中给出一些关于可能的技术发展方向的建议,的确是可以做到的。这也正是本章要达到的目的。

人员: 任何愿意花时间研究软件工程问题的人都可以尝试预测技术的未来发展方向。

重要性: 为什么古代的国王会雇佣占卜者?为什么大多数的跨国公司雇佣咨询公司和智囊团进行预测?为什么相当多的公众相信算命?大家都想知道将要发生什么,以便做好准备。

步骤: 预测前面的路没有什么固定的公

式。我们试图通过收集数据并组织数据以提供有用的信息。为了抽取知识要检查细微的联系,从这些知识中可给出可能趋势的建议,这些趋势可以预测未来会变得怎么样。

工作产品: 对近期的一种看法,可能对,也可能不对。

质量保证措施: 预测未来的道路是一门技艺,不是科学。实际上,绝对正确或错误的严肃预测(幸好,对世界末日的预言例外)都是十分罕见的。我们寻找趋势并尽量推算它们。我们只能随着时间的流逝评估推算的正确性。

在整个软件工程相对短暂的历史中,从业人员和研究人员已经开发了一系列过程模型、技术方法和自动化工具,努力促进计算机软件构建方式的根本变化。尽管过去的经验表明有一个心照不宣的愿望是要找到“银弹”——神奇的过程或卓越的技术,使我们能很轻松地构造大型复杂的基于软件的系统,而没有混淆、没有错误、没有拖延,也没有持续困扰软件工作的众多问题。

但是历史表明寻找银弹好像注定要失败。新技术不断地引入,软件工程师面临的很多问题的“解决方案”被大肆宣传,成为大型或小型项目的一部分。业界权威强调这些“新的”软件技术的重要性,软件界的行家满腔热情地采用这些技术,最终,他们确实在软件工程世界中起了作用,但他们往往并没有履行其承诺。因此,人们的探索还在继续。

在本书过去的版本中(过去 35 年间),我们讨论了一些新技术以及它们对软件工程的预期影响。一些技术已经被广泛采用,但另外一些却从没有达到它的潜能。我们的结论是:技术来来去去,你和我应该探索的真正趋势是软趋势。我们的意思是,软件工程的进展将遵循业务、组织、市场

关键概念

构造块
协同开发
复杂度
意外需求
趋势周期
创新生命周期
模型驱动的开发
开源代码
开放世界软件
后现代设计
需求工程
软趋势
技术方向
技术演变
测试驱动的开发
工具

和文化的趋势。这些趋势导致了技术的变革。

在这一章中，我们将讨论几种软件工程技术方面的趋势，更多的是讨论在商业方面、组织方面、市场方面以及文化方面的一些趋势，这些可能在未来的 10 ~ 20 年对软件工程技术产生很重要的影响。

38.1 技术演变

Ray Kurzweil 在一本神奇的书中提出了一个引人注目的观点：计算（和其他相关的）技术是如何发展的？[Kur05] 他认为技术的发展类似于生物进化，但其增长速度却比生物进化快几个数量级。演进（不管是生物或技术）作为正反馈的结果出现——“从进化过程的一个阶段所产生的更好方法可用于创建下一个阶段”[Kur05]。

21 世纪的“大问题”是：（1）技术怎样才能快速演进？（2）积极反馈有多么重要的影响？（3）必然要发生的变化将有多么深远的意义？

当引入一种成功的新技术时，最初的概念要经历图 38-1 所示的合理且可预言的“创新生命周期”[Gai95]。在突破阶段，有一个问题是公认的，大家不断试图尝试一个可行的解决方案。在某种程度上，有一个解决方案显示了希望。最初的突破性工作在复制阶段获得再生，并获得更广泛的使用。经验会导致经验规则的创造，这些规则支配技术的使用，多次成功导致了广泛使用的理论，它是在自动化阶段由自动化工具创建的。最后，技术成熟并被广泛使用。

提问 当我们考虑技术演变时，“大问题”是什么？

关键点 计算技术正在按指数规律演化，它的增长很快将出现爆炸式趋势。

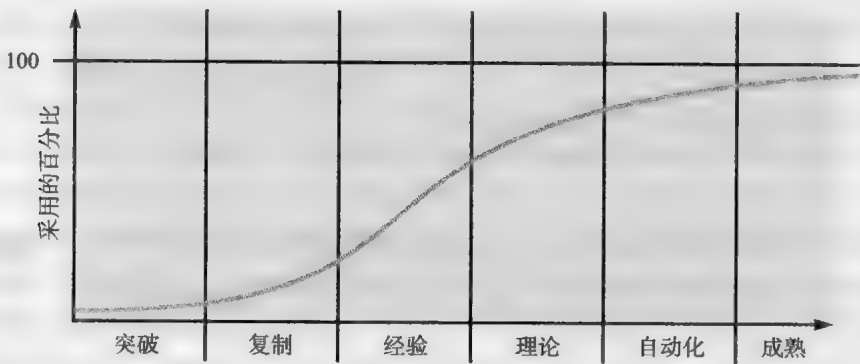


图 38-1 技术演变的生命周期

应该注意到很多研究和技术趋势从来都没有成熟。实际上，绝大多数软件工程领域“有希望的”技术在最近几年得到了广泛关注，然后通过专注的拥护者得到恰当的使用。这一点并不是说这些技术缺乏价值，而是表明通过创新生命周期的途径是漫长而艰难的。

Kurzweil [Kur05] 认为计算技术是以“S 曲线”形式演进的，这表明在技术的形成期增速是相当慢的，在增长期加速度很快，然后随着技术达到极限而达到稳定期。今天，现代计算技术正从 S 曲线的下拐角开始加速发展，即从早期的缓慢增长发展到下一步的爆炸性增长的拐角。这意味着，在未来的 20 ~ 40 年，我们将看到计算能力戏剧性（甚至令人难以置信）的变化。

Kurzweil [Kur05] 认为，在 20 年内，技术演变的加速度将会越来越快，最终导致非生物智慧的时代，它将融合并扩展人类的智慧，这会是迷人的未来。相比之下，无论怎样演

变,所有这一切都需要软件和系统,这些软件和系统会使我们当前的努力显得十分幼稚。到2040年,极值理论、纳米技术、大规模高带宽的普适网络和机器人的结合将带给我们一个完全不同的世界^①。软件可能以我们现在还无法理解的形式继续成为这个新世界的核心。软件工程不会消失。

38.2 关于纯粹工程学科的展望

近50年来,许多学者和业内专家一直提倡建立一个真正的软件工程学科。Mary Shaw在1990年关于这一主题发表了一篇经典的论文,而在其一篇后续论文中[Sha09],她评论道:

工程学科的发展通常从工艺技术的实践中而来,以满足局部或某种特别的使用。当这项技术变得有经济意义时,它便需要稳定的生产工艺和管理控制。由此产生的商业市场是基于经验,而不是基于对技术的深刻理解……当工艺技术变得足够成熟,可以支持有目的的实践和设计可预测的结果时,就出现了工程专业。

841

我们可以证明该行业可以实现“有目的的实践”,但“可预测的结果”一直是不确定的。

随着移动App和WebApp逐渐主宰软件领域,Shaw认为新的挑战存在于“复杂的系统和用户之间的深层依赖关系”[Sha09]。她认为,可以实现“有目的的实践”的知识库已经被特定的社交网络大众化了。举例来说,软件工程师遇到疑问时,不再查询权威编著的软件工程手册,而是将问题发布在论坛里,然后得到很多其他开发者基于经验所给予的回答,而且这些回答经常会有实时的评论,从而可以作为其他的答案以进行选择。

但这不能达到大家要求的学科的水平。Shaw说:“软件工程师面临的问题越来越具有复杂的社会背景,界定问题的边界也是越来越难[Sha09]。因此,分离一门学科的科学基础仍是一个挑战。”此时,在软件领域的发展历史上,这样的描述是合理的,即“截至目前,新软件工程思想的发现是逐渐进化的结果[Erd10]。”

38.3 观察软件工程的发展趋势

Barry Boehm[Boe08]认为:“软件工程师(将)经常面对令人生畏的挑战,处理快速的变化、不确定性和突发事件、可信性、多样性以及互相依赖等问题,但他们还有机会做得更好。”然而,今后几年我们面对这些挑战的趋势是什么呢?

在这一章的引言中,我们提到“软趋势”对软件工程的整个发展方向有重要的影响。但是另外一些(“硬的”)面向研究和技术的趋势依然是重要的。研究趋势“是由技术发展水平和实践发展水平、从业者需要的研究人员看法、能集成特定战略目标的国家拨款的程序以及纯粹的技术兴趣这些笼统的观念驱动的”[Mil00b]。当满足工业界需求和市场机制需要形成的研究趋势被推断出来时,技术趋势就产生了。

38.1节曾讨论了技术演变的S曲线模型。当技术演变时,S曲线适合考虑核心技术的长期影响。但是什么是更适度的、短期的创新、工具和方法呢?Gartner Group[Gar08]——涉及许多行业研究技术发展趋势的顾问

引述 我认为在这个世界上市场中有5台计算机就够了。

Thomas Watson
(IBM 董事长,
1943)

关键点 “趋势周期”表示短期技术集成的现实视图,然而长期趋势是指数级的。

842

^① Kurzweil [Kur05] 提出了合理的技术论据,预测到2029年会出现强大的人工智能(通过图灵测试),并预测人类和机器的演变将会在2045年开始融合。这本书的绝大多数读者会活到那时,看看实际上是否如此。

机构——已经开发了新兴技术的趋势周期，如图 38-2 所示。

并不是每项软件工程技术都要经历这种趋势周期。在某些情况下，幻灭是合理的，技术归于相形见绌。

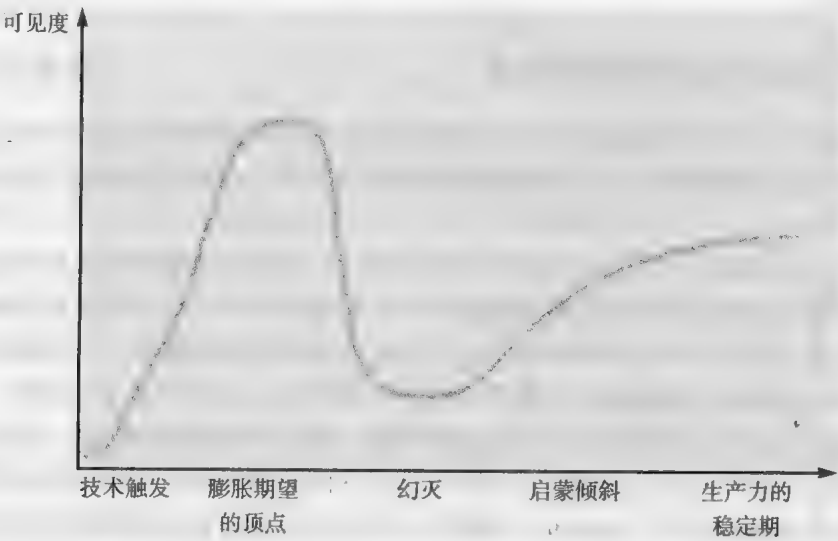


图 38-2 Gartner Group 关于新兴技术的趋势周期 [Gar08]

38.4 识别“软趋势”

每个有着强大 IT 产业的国家都具有独特的特征，这些特征决定了业务运作的方式、公司内呈现的组织动力、面对本地客户的显著销售额以及所有人员互动的强势文化。然而，在这些领域中的一些趋势是普遍的，在社会学、人类学、群体心理学（通常称为“软科学”）方面做的研究与学术界和产业界方面做的研究差不多。

引述 对于任何人，640K 应该足够了。
Bill Gates（微软董事长，1981）

连接和协作（通过高带宽通信做到）可以使软件团队不占用同样的物理空间（实现远程交换和当地条件下的兼职工作）。一个团队可以与位于不同时区且拥有不同主要语言和其他团队协同工作。软件工程必须以贯穿过程模型来应对“分布式团队”，也就是要足够敏捷，以满足即时的需求，而且要遵守纪律以协调不同的群体。

全球化导致了多种多样的劳动力（在语言、文化、问题求解、管理理念、交流偏好以及个人间的相互影响方面都存在差异）。这反过来又需要一个灵活的组织结构。不同的团队（在不同的国家）必须以某种方式对工程问题做出反应，这种方式最好能适应他们独特的需要，同时又促进了某种程度的统一性，使得全球的项目得以进行下去。建议这种类型的组织设置尽量少的管理层次，并且更重视团队级别的决策。这可以获得更大的灵活性，但前提是沟通机制已经建立，使每个团队在任何时候都可以了解项目和技术状况（通过联网的群组软件）。软件工程方法和工具可以帮助实现某种程度的统一性（通过具体的方法和工具可以使团队讲同一种“语言”）。软件过程能为这些方法和工具的实例提供框架。

在世界的某些地区（例如，美国和欧洲），人口正在老化。不可否认的人口统计数字（和文化趋势）意味着许多经验丰富的软件工程师和管理人员在未来十年会离开这个领域。

软件工程界必须采取切实可行的机制保留住这些上了年纪的管理人员和技术人员所具有

的知识(例如,模式(第 16 章)的使用就是向正确的方向迈出的一步),使得未来新一代软件工作者能够获得这些知识。在世界上的其他一些地区,从事软件业的年轻人的数量正在迅速增长。这为铸造软件工程文化提供了机会,而没有 50 年来“陈旧学派”的偏见所带来的负担。

据估计,未来十年将有超过 10 亿的新消费者进入世界市场。消费者在新兴经济体的支出将增加一倍,在 10 年内增长到超过 12 万亿美元 [ATK12]。毫无疑问的是,这个非同一般的支出比例将适用于那些由数码组件构成的产品和服务,这些都是基于软件或软件驱动的。这将表明对新软件的需求在日益增长。接下来的问题是,“能开发出新的软件工程技术来满足这一全球性需求吗?”现代市场趋势通常是由供方驱动的^①。另一种情况是需方的需求推动市场。无论哪种情况,在某种程度上有时很难确定创新周期和需求进展哪个是领先的!

最后,人类文化本身将会影响软件工程的方向。每代人都建立了具有自己烙印的本土文化,任何人也不会例外。Faith Popcorn [Pop08],(一个专门研究文化趋势的著名顾问),总结了下面一些特征:“我们的趋势不是时尚的,而是持久的、发展变化的。它们代表了根本的力量,其首要原因是人类的基本需求、态度和愿望。它们帮助我们游览世界,了解正在发生的事情及其原因,并且为未来做好准备。”关于将会对软件工程产生影响的现代文化趋势的详细讨论,最好留给那些“软科学”的专业人士。

844

38.4.1 管理复杂性

写本书的第 1 版时(1982 年),今天我们熟知的这些数码消费产品还不存在,包含上百万行代码(LOC)的基于主机的系统在当时已属于大型系统。今天,小型数码设备包含 6 万~20 万行代码的客户软件并配有几百万行代码的操作系统,这种情况已经很常见了。现代计算机系统包含 1000 万到 5000 万行代码也是很常见的^②。在不远的将来,需要超过 1 亿行代码的系统^③将开始出现^④。

想想那一刻!

考虑具有 10 亿行代码的系统,其接口既要连接外部的世界、其他互操作的系统、互联网(或其继承者),又要连接内部几百万个必须共同工作的构件,使得这个计算巨人成功运行。有没有一个可靠的方法,以确保所有这些连接都允许信息正确流动呢?

考虑项目本身。我们怎样管理工作流并跟踪进展?传统方法的规模是否能上升几个数量级?

考虑工作人员的数量(和他们的工作地点)、人员和技术的协调、无情的变化、多平台的可能性以及多操作系统环境。有没有一种方法来管理和协调在一个大型项目中工作的人员呢?

考虑工程的挑战。我们如何分析成千上万的需求、约束和限制,确保能够发现和纠正不

① 供方采用了“构造它,消费者就会来”的方式进入市场。有时候,一旦发明了独特的技术,消费者就会趋之若鹜。

② 例如,现代 PC 操作系统(例如, Linux、MacOS 以及 Windows)都有 3000 万到 6000 万行代码。移动设备的操作系统软件也超过了 200 万行代码。

③ 在现实中,这个“系统”实际上是具有很多系统的系统——为了取得某个总体目标,数百个互操作的应用系统一起工作。

④ 并非所有复杂的系统都是大型系统。一个相当小的应用(如小于 10 万行代码)也可能相当复杂。

一致性和不确定性、遗漏和错误？我们如何才能设计出一个强大的体系结构足以处理这种规模的系统呢？软件工程师如何才能建立一个变更管理系统，可以处理成千上万的变更呢？

考虑质量保证的挑战。我们如何才能以一种有意义的方式进行确认和验证呢？又怎么测试一个 10 亿行代码的系统？

在早期，软件工程师试图以一种特别的方式管理复杂性。今天，我们使用过程、方法和工具以保持复杂性是可控的。但是明天呢？我们当前的方法能胜任这些任务吗？

引述 任何人都希望家里有电脑，这不需要什么理由。

Ken Olson (数字设备公司创始人、总裁、董事长, 1977)

38.4.2 开放世界软件

诸如情境智能^①、上下文敏感应用以及普适计算等概念都将集中在把基于软件的系统集成到比个人计算机、移动计算设备或任何其他数字设备广泛得多的环境中。这些关于计算的不远未来的不同观点都集中提出了“开放世界软件”的概念，这种软件“通过自组织结构和自适应行为”来适应不断变化的环境 [Bar06b]。

关键点 开放世界软件包括情境智能、上下文敏感应用和普适计算。

为了有助于说明软件工程师在可预见的未来所要面对的挑战，考虑一下情境智能 (ambient Intelligence, aml) 这个概念。Ducatel [Duc01] 定义情境智能如下：“人们被嵌入很多种物体上的智能的和直观的界面所包围。情境智能环境能够以一种无缝、无障碍方式识别和响应不同个体（在工作时）的存在。”

随着低成本、强功能的智能手机的广泛使用，aml 系统正变得无处不在。软件工程师面临的挑战是如何开发出这样一款应用，它既能满足用户不断增长的新功能需求，同时也适用于各类产品，而且这些功能同时也能保护我们的隐私。

38.4.3 意外需求

在软件项目开始的时候，有一个同样适用于每个利益相关者的老生常谈的话题：“你不知道自己不知道什么。”这意味着，客户很少定义“稳定的”需求。也意味着，软件工程师也不可能总是预见哪里含糊不清以及哪里是矛盾所在。需求变更本来就不是什么新鲜事。

由于系统变得越来越复杂，因此，即使是对陈述全面需求的初步尝试也是注定要失败的。总体目标的陈述是可能的，中间目标的描述也可以实现，但稳定的需求——却不可能！随着每个参与复杂系统的工程设计和建设的人对系统本身、系统所处环境和与系统交互的用户具有了更多的了解，需求就出现了。

建议 由于现实中存在很多突发的需求，因此你的组织应该考虑采取渐进过程模型。

这一现实暗示了很多软件工程趋势。首先，过程模型的设计必须包含变更，并采取敏捷哲学的基本原理（第 5 章）。其次，必须明智地使用产生工程模型的方法（例如，需求模型和设计模型），因为这些模型随着更多知识的获取将不断发生改变。最后，支持过程和方法的工具必须很容易适应和改变。

但对于意外需求还有另一个方面。今天绝大多数的软件开发都假设软件系统和外部环境之间存在的边界是稳定的。边界可能会改变，但这种改变会以一种受控制的方式进行，也

① 关于情境智能的有价值且相当详细的介绍可以在 www.emergingcommunication.com/volume6.html 找到。可在 www.ambientintelligence.org/ 获得更多信息。

就是把软件作为一个普通的软件维护周期的一部分进行调整。这个假设正在开始改变。开放世界软件（38.2.2 节）要求计算机系统“动态地适应和响应变更，即使这些变更是意料之外的”[Bar06]。

就其性质而言，意外需求带来转变。我们如何控制广泛使用的应用程序或系统在其生命周期中发生的演变，并且这对我们设计软件的方式有什么影响呢？

随着变更数量的不断增多，意料之外的副作用出现的可能性也增大了。这应该是为具有意外需求的复杂系统考虑规范的原因。软件工程界必须开发一些方法，帮助软件开发团队预测变更对整个系统的影响，从而减轻意料之外的副作用。今天，我们做到这一点的能力是极为有限的。

38.4.4 人才技能结合

随着基于软件的系统变得越来越复杂，随着全球异地团队之间的通信和协作变得越来越普遍，随着意外需求（产生变更流）变得越来越规范，软件工程团队的真正性质可能发生变化。作为各种复杂系统的一部分，每个软件团队必须拥有创新的人才和技术技能，整个过程必须允许这些人才孤岛的工作结果实现有效的合并。

Alexandra Weber Morales [Mor05] 提出了人才组合的“软件梦之队”。大脑是总设计师，他能够掌控利益相关者的需求，并将这些需求映射到一个既可扩展又可实现的技术框架中。Data Grrl 是一个数据库和数据结构大师，他“通过对谓词逻辑和集合论的深刻理解，将其关连到关系模型，大量使用行和列”。Blocker 是一位技术领导者（经理），他允许团队自由地工作，不受其他团队的影响，同时确保合作的进行。Hacker 是一位出色的程序员，他在家工作，可同时有效地使用模式和语言。Gatherer “灵巧地发现系统的需求，具有……人类的洞察力”，并能清晰准确地表达出来。

38.4.5 软件构造块

促进软件工程哲学的所有人都重视重用的必要性——源代码、面向对象的类、构件、模式以及商业成品软件（COTS）的重用。虽然软件工程界已经取得进展，试图捕获过去的知识以及重复使用可靠的解决方案，但是当今制造的软件很大一部分仍然是“从零开始”。部分原因是利益相关者和软件工程人员对“独特解决方案”的持续渴望。

在硬件界，数码产品的原始设备制造商（Original Equipment Manufacturer, OEM）使用几乎完全由芯片厂商生产的特定应用的标准产品（Application-Specific Standard Product, ASSP）。这种“商业硬件”提供了实现任何数码产品（从智能机到便携式计算设备）所必需的构造块。越来越多的 OEM 厂商使用“商业软件”——专为独特的应用领域（例如 VoIP 设备）设计的软件构造块。Michael Ward [war07] 评论道：

使用软件构件的一个优点是 OEM 可以提升软件所提供的功能，而无需具备特定功能的内部开发经验，也无需将开发人员的时间投入到实施和验证这些构件之中。其他优点包括只需要获取和部署系统需要的一组特定功能的能力，以及将这些构件集成到已有体系结构中的能力。

除了作为商业软件的构件包，越来越倾向于采用软件平台解决方案，这种解决方案“将相关功能的集合合并在一起，这些功能通常在一个集成的软件框架中提供”[War07]。一个软件平台不受一起工作的开发基本功能的 OEM 的影响，而且允许 OEM 将软件工作集中在

有别于其产品的那些特性上。

38.4.6 对“价值”认识的转变

在 20 世纪最后的 25 年，在讨论软件时，商务人士通常问的问题是：“为什么它值那么多钱？”这个问题今天已经很少有人再问了，取而代之的是：“为什么我不能快点得到它（软件和基于软件的产品）？”

在考虑计算机软件时，对价值的最时髦认识已经从商业价值（价格和收益率）向客户价值转变，包括交付的速度、功能的丰富性以及总体产品质量。

38.4.7 开源

谁拥有你或你的组织所使用的软件？日益增加的答案是“每个人”。对“开源”运动的描述如下 [OSO12]：“开源是一种软件开发方法，运用了分布的同行评审的力量和过程的透明性。开放源码的承诺是更好的质量、更高的可靠性、更大的灵活性、更低的成本，而且也是垄断性厂商的坟墓。”开源这个术语应用到计算机软件，意味着对于软件工程的工作产品（模型、源代码、测试套件）都是公开的，任何感兴趣并得到允许的人都可以（有控制地）对其进行评审和扩展。

如果您有进一步的兴趣，Weber [Web05] 提供了一个有价值的介绍，Feller 和他的同事 [Fel07] 已编辑了一本全面、客观的文集，其中考虑了与开源相关的利益和问题，Brown [Bro12] 提供了更多的技术讨论。

引述 但是，好处究竟是什么呢？

IBM 高级计算机系统部的工程师就芯片发表的评价

38.5 技术方向

我们总是认为，软件工程似乎比它的实际变化更快。本节将介绍一个新的“广泛宣传的”技术（它可能是一个新的过程、一个独特的方法或者一个令人兴奋的工具），专家们认为“一切”都会改变。但是，软件工程不只是技术——软件工程是有关人以及交流他们的需求、不断创新、使这些需求成为现实的能力。每当有人参与，变化就会时断时续地慢慢发生。只有达到了一个“转折点” [Gla02]，跨越整个软件工程界的技术阶梯和基础广泛的变化才会真正发生。

在这一节中，我们将讨论在过程、方法和工具几个方面的一些趋势，这些可能会对未来十年的软件工程产生影响。它们会导致转折点出现吗？我们将拭目以待。

引述 对待数字技术正确的、带有艺术性的反应是把它当作一切事物的新窗口，并以激情、智慧、无畏和喜悦的态度为人类所永恒使用。

Ralph Lombreglia

38.5.1 过程趋势

可以说，所有的商业、组织和 38.4 节中讨论的文化趋势都增强了对过程的需要。但是第 37 章讨论的框架提供了通向未来的路线图吗？过程框架将会演进，它能在纪律性和创造性之间找到更好的平衡吗？软件过程会适应那些采购软件的、构建软件的和使用软件的利益相关者的不同要求吗？它能否提供一个同时为这三组人减少风险的手段？

这些以及其他许多问题依然悬而未决。

在下面的段落中，我们来介绍 Conradi 和 Fuggetta [Con02] 提出的 6 点可能的过程趋势。

1. 随着 SPI 框架的发展, 它将强调“关注目标定位和产品创新的策略” [Con02]。在软件开发的快节奏的世界里, 长期的 SPI 策略很少在动态的商业环境中生存下来。太多的变化来得太快。这意味着, 稳定的、按部就班的 SPI 路线图可能被强调短期产品定位目标的框架所取代。
2. 因为软件工程师对一个过程中哪里存在弱点是比较清楚的, 所以过程改变一般应按他们的要求驱动, 并且应该自底向上进行。Conradi 和 Fuggetta [Con02] 建议未来的 SPI 活动应该“以一种简单的、聚焦的积分卡开始, 而不是大规模的评估”。通过严密地关注 SPI 工作以及自底向上的工作, 业务人员将能看到早期实质性的变化——在软件工程工作中产生了真正的差别。
3. 自动的软件过程技术 (Software Process Technology, SPT) 将不做全局性过程管理 (整个软件过程的广泛支持), 而是侧重于软件开发过程中从自动化中最能受益的那些方面。没有人会反对工具和自动化, 但在很多情况下, SPT 并没有履行承诺 (38.3 节)。要想最有效, 应侧重于普适性活动 (第 3 章), 即软件过程中最稳定的元素。
4. 更强调 SPI 活动的投资收益率。在第 37 章, 我们已经学习了投资收益率 (ROI) 定义如下:

$$ROI = \frac{\sum(\text{收益}) - \sum(\text{成本})}{\sum(\text{成本})} \times 100\%$$

迄今为止, 软件组织一直在努力以量化方式明确界定“收益”。可以说 [Con02] “我们因此需要一个标准化的市场价值模型来解释软件改进”。

5. 随着时间的推移, 软件界可能逐渐认识到, 如其他技术性更强的学科一样, 社会学和人类学的专业知识对于成功的 SPI 有很多或更多的事情可做。除此之外, SPI 改变着组织的文化, 文化的改变涉及个人和群体。Conradi 和 Fuggetta [Con02] 正确地谈道: “软件开发人员是知识工作者。他们往往对高层就如何做工作或改变过程的指挥反应消极。”可以通过考察群体社会学学到更多的知识, 更好地了解引进变革的有效方法。
6. 学习的新模式可能有助于向更有效的软件过程转变。在这种情况下, “学习”是指从成功和错误中学习。收集度量 (第 30 章和 32 章) 的软件组织明白过程因素将如何影响最终产品的质量。

38.5.2 巨大的挑战

有一个趋势是不可否认的——毋庸置疑, 随着时间的推移, 基于软件的系统将变得越来越大、越来越复杂。正是这些复杂的大型系统工程, 无论交付平台还是应用领域, 都构成了对软件工程师的“巨大挑战” [Bro06]。Manfred Broy [Bro06] 建议软件工程师通过创建新的方法理解系统模型, 并利用这些模型作为基础构建高质量的下一代软件以满足“复杂软件系统开发的严峻挑战”。

随着软件工程界不断开发出新的模型驱动方法 (在这节的后面讨论) 来表示系统需求和设计, 下面的一些特征 [Bro06] 值得关注。

- 多功能性——随着数码产品的进化, 它们已开始提供一套内容丰富且有时互不相关的功能。移动电话一度被认为是简单的通信设备, 现在已成为功能很强的袖珍计算机, 可以完成比通话更为重要的功能。正如 Broy [Bro06] 指出的, “工程师必须详细描述所交付功能

提问 未来十年最可能的过程趋势是什么?

849

提问 对于未来的应用, 分析师和设计师必须考虑的系统特性是什么?

850

的背景，而最重要的是，必须确定系统的不同特性之间可能有害的相互作用”。

- 反应性和及时性——数码产品与现实世界的互动日益增多，必须对外部刺激做出及时反应。它们必须与各种各样的传感器接口连接，必须在与当前任务相适应的时间框架内做出反应。因此必须开发新方法：（1）帮助软件工程师预测各种反应特性的时间；（2）以一种使功能较少依赖于机器并且更便携的方式实现这些特征。
- 用户交互的新方式——软件的开放世界趋势意味着必须对新的交互方式进行建模和实现。无论这些新方法是否使用触控界面、语音识别或直接智能接口[⊖]，数码产品的新一代软件都必须适应他们。
- 复杂的体系结构——拥有超过 2000 个功能的豪华汽车已经由软件控制，包含在复杂的硬件架构中，包括多处理器、先进的总线结构、执行器、传感器、日趋复杂的人机界面和许多安全相关的组件。更复杂的系统就在眼前，这对软件设计师提出了重大的挑战。
- 异构的分布式系统——任何现代嵌入式系统的实时组件都可以经过内部总线、无线网络或者因特网（或者所有这三者）连接起来。
- 关键性——在几乎所有的商业关键系统和大多数安全关键系统中，软件已经成为其中的中枢组件。然而，软件工程界仅仅开始应用软件安全的最基本原理。
- 维护可变性——在一个数码产品中，软件的寿命很少能超过 3～5 年，但是，飞机中复杂的航空系统至少可以使用 20 年。汽车软件介于两者之间。这些对设计有影响吗？

Broy [Bro06] 认为，只有在软件工程领域开发出更有效的分布式和协作软件工程的理念、更好的需求工程方法、更健壮的模型驱动的开发方法以及更好的软件工具，这些软件特点和其他软件特点才能成为可控制的。接下来的几节中将简要地探讨这些领域。

38.5.3 协同开发

软件工程是一门信息技术学科。这一点虽然是显而易见的，但我们还是要强调。从任何软件项目开始，每个利益相关者都必须共享信息——有关基本的商业目标和目的，有关特定系统的需求，有关体系结构的设计问题，有关软件构造的几乎每个方面。

关键点 协作涉及信息的及时传播以及沟通和决策的有效过程。

今天，软件工程师可以跨越时区和国界进行合作，他们中的每个人都要共享信息。这同样适用于开源项目，其中数百或数千名软件开发人员一起工作以建立一个开源应用程序。同样，信息必须传播，以保证开放的协作成为可能。

38.5.4 需求工程

在第 8～11 章中已经介绍了基本的需求工程活动——需求获取、细化、协商、规格说明和确认。这些活动的成功和失败对于整个软件工程过程的成功和失败有着重要的影响。然而，需求工程（Requirements Engineering, RE）已被人们比作“试图将软管夹夹在冰糕上”[Gon04]。正如本书很多地方都提到的，软件需求有不断变更的趋势，并且随着开放世界系统的出现，意外需求（和近乎持续的变更）可能会变得普遍。

⊖ 直接智能接口的简单讨论可以在 http://en.wikipedia.org/wiki/Brain-computer_interface 找到，在 <http://au.gamespot.com/news/6166959.html> 上描述了一个商业实例。

今天,绝大多数“非形式化的”需求工程方法以创建用户场景(例如用例)开始。更形式化的方法创建一个或更多的需求模型,并以此作为设计的基础。形式化方法能使软件工程师通过使用可验证的数学符号来表示需求。当需求稳定时,所有这一切都可以合理地工作,但对于动态需求或意外需求问题就不容易解决了。

有许多不同的需求工程研究方向,包括:从翻译文本描述到更结构化的表示(例如分析类)的自然语言处理;为了结构化和理解软件需求需要更多地依赖数据库;当执行需求工程任务时,使用 RE 模式来描述典型的问题和解决方案以及面向目标的需求工程。然而,在行业层面,RE 活动还停留在非正式和令人不可思议的基础阶段。为了改善需求定义的方式,当执行 RE 时,软件工程界会实施三种不同的子过程 [Gli07]:(1)改进知识获取和知识共享方式,从而更完整地理解应用领域的限制和利益相关者的需求;(2)在定义需求时,更加强调迭代;(3)使用更有效的沟通和协调工具,使所有利益相关者进行有效的合作。

对于前面段落中声明的 RE 子过程,只有当这些子过程已恰当地集成到一个不断改进的软件工程方法中时才会成功。随着基于模式的问题求解和基于构件的解决方案开始支配许多应用领域,RE 必须适应敏捷性(快速增量交付)和由敏捷性所导致的内在的意外需求。静态的“软件规格说明”的概念开始消失,它将被“价值驱动的需求”[Som05]所取代,这是响应利益相关者提出的早期软件增量要求交付的性能和功能的必然产物。

38.5.5 模型驱动的软件开发

几乎在软件工程过程的每一步,软件工程师都使用抽象手段。随着设计的开始,体系结构级和构件级的抽象得到表达和评估。然后它们必须被翻译成一种编程语言表示,即把设计(较高级别的抽象)转换成具有特定的计算环境(较低级别的抽象)的可操作的系统。模型驱动的软件开发^①以某种方式将特定领域的建模语言与转换引擎和产生器相结合,有助于对较高层次抽象的表示,并将其转换成较低层次的表示 [Sch06]。

特定领域的建模语言(DSML)描述“应用程序结构、行为和特定应用领域的需求”并使用元模型进行描述。元模型“定义在领域中概念之间的关系,准确地描述关键语义以及与这些领域概念相关的约束”[Sch06]。DSML 与通用的建模语言(如 UML 见附录 1)的主要区别是 DSML 协调应用领域的设计概念,因此可以以一种有效率的方式表示设计要素之间的关系和约束。

关键点 模型驱动方法定位于所有软件开发者面临的挑战——怎样在一个比代码级抽象程度更高的级别上表示软件。

853

38.5.6 后现代设计

在一篇有趣的“后现代时代”软件设计的文章中,Philippe Kruchten [Kru05]给出了以下观点:

计算机科学还没有达到可以解释一切大境界的程度,宏观上我们没有发现像在物理或其他工程学科中发挥作用的基本规律一样的软件基本规律。我们仍然生活在互联网泡沫破灭和 Y2K 数字末日的苦涩回味中。因此,在这个后现代的时代,这里发生的一切事情似乎都有点重要,却又不是真正的重要,软件设计未来的方向是什么呢?

任何企图了解软件设计趋势的尝试都是要建立设计的边界。需求工程在哪里停止?设计从哪里开始又在哪里停止?从哪里开始生成代码?对这些问题的回答可能并不像看起来那么

① 术语模型驱动工程(Model-Driven Engineering, MDE)也会使用。

容易。即使需求模型应侧重于“做什么”而不是“如何做”，但每个分析师都做了一点设计，而几乎所有的设计师都做了一些分析。同样，随着软件构件的设计更接近算法细节，设计师开始以更接近代码级的抽象表示构件。

38.5.7 测试驱动的开发

需求驱动设计，设计成为构建的基础。简单的软件工程实际运作过程是相当起作用的，对于创建软件体系结构不可或缺。然而，当考虑构件级的设计和构建时，巧妙的变更可以提供极大的好处。

在测试驱动的开发（Test-Driven Development, TDD）中，将软件构件的需求作为生成一组测试用例的基础，以测试用例检查接口，并试图找到数据结构中和构件提供的功能中的错误。TDD 并不是一种真正的新技术，而是一个趋势，它强调生成源代码之前的测试用例的设计[⊖]。

TDD 过程遵循简单的流程，如图 38-3 所示。在第一小部分的代码生成之前，软件工程师引入测试以备检测代码（尽量发现代码的错误）。然后写下的代码要顺应测试。如果通过了，再创建新的测试来检测要开发的下一段代码。此过程持续下去，直到各构件完全编好代码，并且所有的测试都没有错误地执行了。然而，如果任何测试成功地发现了错误，那么现有的代码就要重构（修正），所有和错误相关的测试都重新执行。这种重复的流程继续进行，直到没有测试需要创建，这意味着该构件符合定义的所有要求。

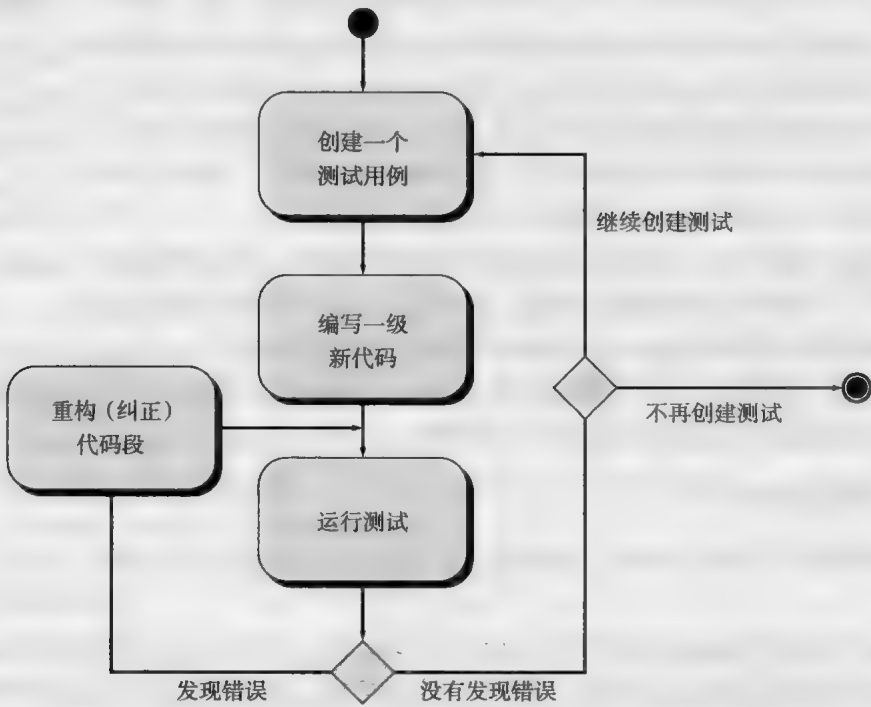


图 38-3 测试驱动的开发过程流

在 TDD 中，代码是以非常小的增量（一次一个子函数）开发的，直到存在测试要检查时，才编写代码。应该注意到，每个迭代都会产生一个或多个新的测试并添加到回归测试集

⊖ 回忆一下极限编程（第 3 章），它强调这种方法是敏捷过程模型的一部分。

854

中，每次变更都要运行回归测试集。这样做是为了确保新代码没有在旧代码中产生导致错误的副作用。如果对 TDD 有进一步的兴趣，请参见 [Bec04b]、[Ste10] 或 [Whi12]。

38.6 相关工具的趋势

每年都有数以百计的工业级软件工程工具被采用。其中大多数是由工具制造商提供的，他们声称这些工具会改进项目管理、需求分析、设计建模、代码生成、测试、变更管理或本书讨论的任何软件工程活动、行为和任务。其他工具则已经作为开源产品被开发出来。大多数开源工具都特别强调构建活动（特别是代码生成）中的“编程”活动。还有一些工具来自于大学和政府实验室的研究工作。虽然他们在非常有限的应用问题中很吸引人，但是大多数不具备广泛的行业应用。

在行业层面，最全面的工具包形成了软件工程环境（Software Engineering Environment, SEE）^①，它围绕一个中心数据库（存储库）集成了特定的工具集。当作为一个整体考虑时，SEE 通过软件过程整合了信息，并且为许多大型且复杂的基于软件的系统所需的协作提供帮助。但当前的环境是不容易扩展的（很难集成 COTS 工具，它不是软件包的一部分），而且往往是通用的（即它们不是针对具体的应用领域）。新技术解决方案（例如，模型驱动软件开发）的推出和切实可行的支持新技术 SEE 的可用性之间也存在一个很长的时间间隔。

软件工具的未来趋势将遵循两种不同的途径：一种是以人为本的途径，对应了 38.4 节中讨论的“软趋势”；另一种是以技术为中心的途径，关注新技术的引进和采用（38.5 节）。

38.4 节中讨论的软趋势——需要管理复杂性、满足意外需求、建立包含变更的过程模型、协调具有不断变化的人才组合的全球团队等——预示着一个新的时代，在这个时代，支持利益相关者协作的工具将变得和支持技术的工具同样重要。

当利益相关者作为一个团队工作时，敏捷软件工程（第 5 章）是可以实现的。因此，即使各自在本地开发软件，面向协同 SEE 的趋势也是有益的。哪些技术工具可以补充系统和构件，使得软件工程师能够更好地协作？

技术工具的主要趋势之一是建立工具集，支持突出体系结构驱动设计的模型驱动开发（38.5.5 节）。Oren Novotny [Nov04] 建议，模型应成为软件工程的中心焦点，而不是源代码：

可以使用 UML 建立平台独立的模型，然后进行不同程度的改造，最终形成一个特定平台的源代码。那么，不言而喻，是这个模型——而不是文件——应该成为新的输出单元。模型在不同的抽象层次上会有很多不同的视图。在最高层次，可以确定分析中平台独立的构件；在最低层次，特定平台的实现可以分解到代码形式的一组类上。

Novotny 认为，新一代工具将与存储库联合工作，在所有需要的抽象层次上创建模型，建立不同模型之间的关系，从其中一个抽象层次的模型转换到另一个层次的模型（例如，将设计模型转换成源代码），管理变更和版本，针对软件模型协调质量控制和质量保证行动。

除了完整的软件工程环境，关注从收集需求到设计 / 代码重构再到测试的任何事情的单点解决方案工具将继续发展，并提供更多的功能。在有些情况下，与通用工具相比，针对特定应用领域的工具将提供更多的好处。

① 也使用术语集成开发环境（Integrated Development Environment, IDE）。

38.7 小结

对软件工程技术有影响的趋势经常来自于商业、团体、市场和文化等领域。这些“软趋势”能指导研究方向以及作为研究结果的技术。

引入的每项新技术都会经历一个生命周期，新技术并不总是被广泛采用，即使最初的期望很高。任何软件工程技术获得广泛应用的程度都与解决软趋势和硬趋势问题的能力相关联。

软趋势——对连接和协作、全球项目、知识转让、新兴经济的影响以及人类文化本身的影响力等不断增长的需要，导致了跨越管理复杂性和意外需求的一系列挑战，需要调整分散在各地的软件开发团队不断变化的人才结构。

硬趋势——技术变化的步伐在日益加快——决定软趋势，并影响软件结构、过程范围和表示过程框架特性方式。协同开发、需求工程的新形式、基于模型的开发和测试驱动的开发以及后现代设计会改变方法的前景。对于不断增长的沟通与协作需要，工具环境将做出回应，同时集成了特定领域点的解决方案，这些可能会改变目前软件工程任务的性质。

习题与思考题

- 38.1 阅读 Malcolm Gladwell 的畅销书《The Tipping Point》，讨论如何将他的理论应用于新软件工程技术。
- 38.2 为什么开放世界软件对传统的软件工程方法提出了挑战？
- 38.3 回顾 Gartner Group 关于新兴技术的趋势周期。选择一个众所周知的技术产品，并给出它的简要发展史，说明它是如何沿着曲线发展的。选择另一个著名的技术产品，但它不遵循趋势周期的发展。
- 38.4 什么是“软趋势”？
- 38.5 当你面对着一个极其复杂的问题时，这需要一个漫长的解决方案。你如何去处理这种复杂性并给出巧妙的解决方案？
- 38.6 什么是“意外需求”，它们为什么对软件工程师提出了挑战？
- 38.7 选择一个开源的开发工作（除了 Linux），并提交其演变和相对成功的简要发展史。
- 38.8 描述你认为软件过程在未来十年将会怎样改变？
- 38.9 你在洛杉矶并且和全球软件工程团队一起工作。你和在伦敦、孟买、中国香港和悉尼的同事必须为一个大系统编辑 245 页的需求规格说明。必须在 3 天内完成初稿。描述一套理想的在线工具，可以使你们能够有效合作。
- 38.10 用自己的话描述模型驱动的开发及测试驱动的开发。

扩展阅读与信息资源

讨论软件和计算发展的书籍涉及大量的技术、科学、经济、政治和社会问题。Kurzweil (《The Singularity Is Near》，Penguin Books, 2005 和《How to Create a mind》，Viking, 2012) 提出了一个引人注目的观点，到本世纪中叶，我们对世界的看法将发生真正深刻的变化。Sterling (《Tomorrow Now》，Random House, 2002) 提示我们：真正的进展是很少有秩序和有效率的。Nanz (《The Future of Software Engineering》，Springer, 2010) 以及 Draheim 和他的同事 (《Software Engineering Tools : Trends of Software Engineering Tools and Platforms》，2010) 讨论了软件开发的趋势。Meisel (《The Software Society : Culture and Economic Impact》，Trafford, 2013)、Saylor (《Mobile Wave: How

Mobile Intelligence Will Change Everything》, Vanguard Press, 2012), Dourish 和 Bell(《Divining a digital future: mass and Mythology in Ubiquitous Computing》, MIT Press, 2011) 以及 Teich(《Technology and the Future》, 12th ed, Wadworth, 2012) 对有影响力的技术以及改变文化状态的技术给出了详细的论述。Philis 和 Naisbitt(《High Tech/High Tough》, Nicholas Brealey, 2001) 指出, 我们许多人都“陶醉于”高科技, 并且“在高科技时代的巨大讽刺是, 我们已经成为了设想给我们自由的设备的奴隶”。Zey(《The Future Factor》, Transaction Publisher, 2004) 讨论了将对本世纪人类的命运形成重大影响的力量。Negroponte 的书(《Being Digital》, Alfred A. Knopf, 1995) 是 20 世纪 90 年代中期的畅销书, 并继续提供对计算及其全面影响的独到见解。

随着软件成为我们生活中几乎每个方面的一部分, “网络空间伦理”已经演变为一个重要的讨论话题。Quninn(《Ethics for the Information AGE》, 5th ed. Addison-Wesley, 2012), Spinello(《Cyberethics: Morality and Law in Cyberspace》, 4th ed., Jones & Bartlett Publishers, 2010), Tavini(《Ethics and Technology》, 3rd ed., Wiley, 2010), Halbert 和 Ingulli(《Cyberethics》, South-Western College Publishers, 2004) 以及 Baird 和他的同事(《Cyberethics: Social and Moral Issue in the Computer Age》, Prometheus Books, 2000) 详细考虑了这个主题。美国政府以 CD 光盘的形式发布了一份长篇报告(《21st Century Guide to Cybercrime》, Progressive Management, 2003), 考虑了计算机犯罪、知识产权问题以及国家基础设施保护中心(National Infrastructure Protection Center, NIPC)的所有方面。

关于软件技术和软件工程未来方向的大量信息可以在网上找到。最新的有关软件工程未来发展趋势的参考文献可在 SEPA 网站 www.mhhe.com/pressman 上找到。

结 束 语

要点浏览

概念: 当我们即将结束这个关于软件工程的相对漫长的旅程时, 该阐明一些观点并提出一些结论性意见了。

人员: 与本书作者一样的人。当你即将看完这本漫长的、富有挑战性的书的时候, 很高兴能以一种有意义的方式做个总结。

重要性: 记住我们曾经所处的位置并考虑我们要达到的目标总是有价值的。

步骤: 我将会考虑我们所处的位置和要解

决的一些核心问题以及未来的一些发展方向。

工作产品: 帮助你理解全局的讨论。

质量保证措施: 立竿见影是很难的。只有经过若干年以后, 我们才能够分辨出本书中讨论的这些软件工程的**概念、原理、方法和技术**是否已经帮助你成为了更好的软件工程师。

在前面的 38 章中, 我们已经探讨了软件工程过程, 包括管理规程和技术方法、基本概念和原理、专门技术、面向人员的活动和适合于自动化的任务、用纸和笔表示的符号以及软件工具。我认为, 对敏捷度和质量的测量、规定及高度重视将得到满足客户需要的、可靠的、可维护的、更好的软件。但是, 我从未允诺软件工程是万能的。

软件和系统技术对于构造计算机系统的软件专业人员 and 公司仍是一种挑战。虽然 Max Hopper [Hop90] 是怀着对 21 世纪的展望写下了下面这些话的, 但却准确地描述了当前的情形:

因为信息技术的变化正变得如此快速和不可遏制, 并且落后的后果是如此的不可挽回, 因此, 公司要么掌握技术, 要么倒闭……细想起来对技术“是多么无奈”, 公司不得不拼命地追赶, 才能有立足之地。

软件工程技术方面的变化确实“快速和不可遏制”, 但同时进展又经常很慢。在决定采用一种新过程、方法或一种新工具的时候, 为了理解其应用, 需要进行必要的培训, 然后将技术引入软件开发文化中, 此时会伴随出现某些新东西 (以及更好的东西), 而且过程也将重新开始。

在这个领域中, 有一件事我琢磨了多年, 就是软件工程从业人员是“赶时髦”的人。前面的道路将充满令人兴奋的新技术 (最新的潮流), 这些技术还从来没有真正使用过 (尽管进行了大量宣传)。这将形成更合适的技术, 以某种方式修改前进道路的方向和宽度。其中的少数内容我在第 38 章已经讨论了。

在这最后一章, 我将从更哲学的角度扩展我的观点, 并进一步考虑在软件工程实践领域

关键概念

人工智能沟通
道德规范
未来
信息范围
知识
人员
职责
软件回顾

我们今天所处的位置以及我们的目标。

39.1 再论软件的重要性

可以从很多方面来叙述计算机软件的重要性。在第1章中，软件被描述为区分器。交付的软件可区分为产品、系统和服务，它提供了市场竞争力。但是，软件并不仅仅是区分器。当从整体上考虑的时候，软件工程的工作产品产生了任何个人、商业或政府都能获取的最重要的日用品——信息。

在第38章中，我简单地讨论了开放世界的计算将从根本上改变我们对计算机的看法、我们利用计算机所做的事情（和它们为我们所做的事情）以及我们将信息视为一种指导、一种商品甚至一种必需品的认知。我还注意到支持开放世界计算的软件将会给软件工程师提出新的激动人心的挑战。但更为重要的是，未来还在成长并且普遍存在的计算机软件将给整个社会提出更加急剧的挑战。每当技术具有广泛的影响——这种影响可以挽救或危害生命、建立或摧毁多个行业以及引导或误导政府领导人，那就必须要“小心处理”了。

39.2 人员及其构造系统的方式

高科技系统需要的软件每年都变得越来越复杂，最后形成的程序规模也成比例地增长。如果不是因为“随着程序规模的增长，必须为此程序投入的人员数量也要增加”这样一个简单事实的话，“平均”程序规模的快速增长并不会给我们带来太多问题。

经验表明，如果一个软件项目团队的人数增加，则项目团队的整体生产率可能会下降。针对这个问题的一个解决方法就是增加软件工程项目团队的数量，从而将人员划分为单独的工作小组。然而，随着软件工程项目团队数量的增加，他们之间的交流也会变得困难和费时，就像个人之间的交流一样。更糟糕的是，交流（在个人间或项目团队间）趋向于低效，即用了太多的时间，却只能传递很少的信息内容。而且，通常情况是将重要的信息“分裂为破碎的片断”。

如果软件工程界有效地解决了交流的困境，软件工程师的未来之路必定会涉及个人之间及项目团队之间相互交流方式的根本性改变。在第38章中，我们讨论了协同工作环境可以在团队交流方面提供显著的改进。

最后，交流是知识的传递。知识获取（和传递）的方式正在发生深刻改变。随着搜索引擎的日益成熟、社交网络和众包 morph 的工具化，以及 Web 2.0 应用所提供的更好的协同能力，知识转移的速度和质量将成倍增长。

如果历史是一面镜子，那么就可以公正地说人类本身并没有改变。但是，他们交流的方式、工作的环境、获取知识的方式、使用的方法和工具、应用的规则以及软件开发的全部文化都将发生重大而深刻的改变。

引述 未来的冲击（是）消除我们个体感受到的压力和迷惑，方法是使他们在极短的时间内遭遇很多变化。

Alvin Toffler

[861]

SafeHome 结论?

[场景] Doug Miller 的办公室。

经理) 和 Vinod Raman (产品团队成员之一)。

[人物] Doug Miller (SafeHome 软件工程

[对话]

Doug: 我非常高兴,我们在没有太多戏剧性事件的情况下完成了工作。

Vinod (叹了口气,靠在椅子上): 是的,但是项目进展不大。

Doug: 你很惊讶? SafeHome 项目开始的时候,市场部认为做一个桌面应用程序就可以实现,然后……

Vinod (笑着): 然后,紧接着在网络和移动端上。

Doug: 但是我们确实学到了很多。

Vinod: 是的。这些高科技的东西太有意思了,但是软件工程类的东西只允许我们把它做到接近计划。

Doug: 是的。这些都是你们的辛勤工作。客户服务那边怎么样? 质量又如何?

Vinod: 是有几个问题,但都不是致命的。我们已经在处理了。5分钟之后,我将与 Jamie 面谈此事。

Doug: 在你去之前……

Vinod (正在往外走): 我知道,提前做些准备。

Doug: 一种新的传感器已经被开发出来了,非常高端,我们在 SafeHome II 中将采用。

Vinod: SafeHome II?

Doug: 是的, SafeHome II。下周我们开始准备计划。

39.3 表示信息的新模式

在计算机的历史上,用于描述商业界软件开发工作的术语发生了一些微妙的变迁。50年前,术语数据处理是描述计算机在商业中应用的习惯用语。今天,数据处理已经让位于另一个短语——信息技术,它与“数据处理”所指的事情是相同的,但在关注点上有微妙的偏移。它强调的重点不仅仅是大量数据的处理,而是从这些数据中抽取有意义的信息。显然,这才是永久的目标,然而,术语上的改变反映了管理哲学上的更重要的变化。

当我们今天讨论软件应用时,数据、信息和内容这些词反复地出现,我们也在某些人工智能应用中遇到知识一词,但是它的使用相对较少。事实上,没有人在计算机软件应用的范畴内讨论智慧。

数据是未加工的信息——事实集合,它必须经过处理才具有意义。信息是在给定的环境下通过相互关联的事实而得到的。知识是将某个环境中所获得的信息与在另外不同环境中获得的信息相关联。而智慧是从完全不同的知识中推导出的一般性原理。图 39-1 以图表形式表示了“信息”的 4 种视图。

目前所构造的绝大多数软件都是用于处理数据或信息,软件工程师还同样关心处理知识的系统^①。知识是二维的,针对一系列相关和不相关的主题而收集的信息被联结在一起,形成一个事实体系,我们称之为知识。其中的关键是这样一种能力,即应该如何将来自一系列不同源(它们之间可能没有明显的联系)的信息联系起来,并将它们组合在一起,为我们提供某些独特的收益。^②

引述 做好明天工作的最好的前提就是做好今天的工作。

Elbert Hubbard

① 数据挖掘和数据仓库技术的快速发展反映了这种增长趋势。

② 语义 Web (Web 2.0) 允许这种 mashups 的创建,它提供了一种易于获取知识的机制。

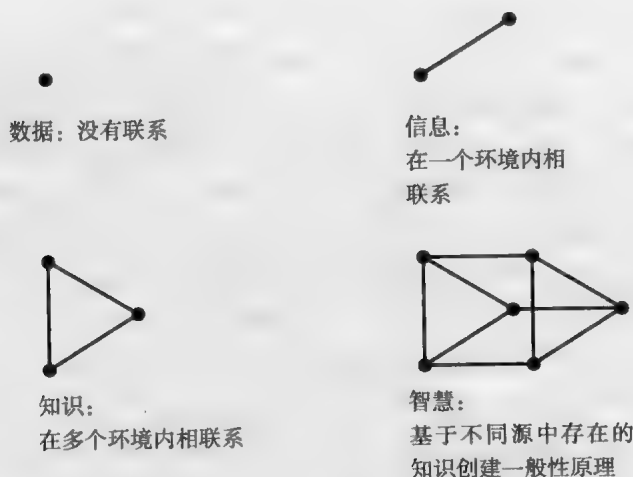


图 39-1 “信息”谱系

为了说明从数据到知识的发展过程，以人口普查数据为例：1996年，美国的出生率是490万，该数字表示一个数据值。将该数据和前40年的出生率相关联，我们可以导出一种有用的信息——正在变老的20世纪50年代及60年代早期的“生育高峰出生的婴儿”正赶在他们的最佳生育年龄结束前做最后的生育努力。另外，“青年一代”（gen-Xers）也已到了生育的年龄。然后，该信息还可与其他似乎无关的信息相关联。例如：将在下一个十年退休的小学教师的当前数量；具有初中和中级教育程度毕业生的数量；或者政治家承受的降低税率的压力；以及因此限制教师薪金增长的压力。

引述 智慧是使我们能够运用知识为自己及他人获得利益的力量。
Thoms J. Watson

可以将这些信息组合起来制定知识的示意图——这在21世纪早期对美国的教育系统来说会是个很大的压力，并且这些压力会持续10余年。利用这些知识会带来一些商业机会，这对于开发新的学习模式来说可能是一个重要的机会，将比现有的方法更有效且成本更低。

软件的未来之路是处理知识的系统。我们使用计算机处理数据已超过50年，抽取信息超过30年。软件工程界面临的最重要的挑战之一是沿着信息谱的发展趋势构造迈向下一步的系统——从数据和信息中以实用、有益的方式抽取知识的系统。

39.4 远景

在39.3节中，我提出未来的道路将通向建立“处理知识”的系统。但是，未来的通用计算和特殊的基于软件的系统可能会导致相当深刻的事件发生。

在一本神奇的涉及计算技术的书中（一定适合每个人阅读），Ray Kurzweil[Kur05]建议，我们已经达到了这样一个时代——“技术变化的步伐是如此之快，影响是如此之深，以至于人们的生活将不可逆转地跟着改变”。Kurzweil^①给出了令人信服的论证，我们目前正处在指数增长曲线的“转折点”，在未来20年中，计算能力将得到极大提升。随着纳米技术、基因技术和机器人技术方面的进展，可能在本世纪中期的某个时候，人类（如同我们知道他们今天的样子）和机器之间的区别将变得很模糊，这个时候人类正以某种既可怕（对一些人来

① 值得注意的是，Kurzweil不是一个磨坊中的科幻作家或没正事的未来主义者。他是一位严肃的技术主义者，“已经成为光学符号识别（Optical Character Recognition, OCR）、文字语音合成、语音识别技术以及电子键盘设备等领域的先驱”（Wikipedia）。

说)又壮观(对另外一些人来说)的方式加速进化。

Kurzweil 认为,在未来十年的某个时候,计算能力和必要的软件将足以对人脑的每个方面进行建模 [Kur13]——所有的物理连接、模拟过程和化学覆盖。这种情况发生时,人类将首先获得“强人工智能”,因此,机器确实可以思考(使用今天世界的常规说法)。但是,存在一个根本的差别。人脑的处理是相当复杂的并且和外部的信息源仅仅以一种松散的方式连接。即使与今天的计算技术相比,人脑的计算也是很慢的。当人类的大脑能完全仿真时,机器的“思想”将以比人脑快数千倍的速度更迅速地与相应的海量信息相连接(作为一个简单的例子,想一想今天的网络)。其结果是……如此神奇,最好是留给 Kurzweil 来叙述。

值得注意的是,不是每个人都相信 Kurzweil 描述的未来,这是一件好事。在著名的题为“The Future Doesn’t Need Us”的论文里, Bill Joy[Joy00](Sun 公司的创始人之一)认为“机器人技术、基因工程、纳米技术正在威胁着人类这个濒临灭绝的物种”。他对技术的不良预测与 Kurzweil 所预言的理想国未来形成了对立。双方都应认真考虑,作为一个软件工程师在确定人类的长远未来时应起什么样的引导作用。

39.5 软件工程师的责任

软件工程已经发展成为令人尊敬的、全球性的行业。作为专业人员,软件工程师应该遵守职业道德规范,以指导他们所做的工作及他们所生产的产品。ACM/IEEE-CS(美国计算机协会/国际电器与电子工程师协会和计算机科学会社)联合工作组已经提出了《Software Engineering Code of Ethics and Professional Practices》(软件工程职业道德规范和职业实践要求)(5.1 版),该规范 [ACM12] 规定:

软件工程师应履行其承诺,使软件的分析、规格说明、设计、开发、测试和维护成为一项有益和受人尊敬的职业。依照他们对公众健康、安全和利益的承诺,软件工程师应当坚持以下八项原则:

1. 公众——软件工程师的行为应符合公众利益。
2. 客户和雇主——在保持与公众利益一致的原则下,软件工程师的行为应使他们的客户和雇主获得最大利益。
3. 产品——软件工程师应该确保他们的产品和相关的修改符合最高的专业标准。
4. 判断——软件工程师应当维护他们职业判断的完整性和独立性。
5. 管理——软件工程师经理和领导应赞成和促进对软件开发和维护进行合乎道德规范的管理。
6. 专业——在与公众利益一致的原则下,软件工程师应当推进其专业的完整性和声誉。
7. 同事——软件工程师对其同事应持平等和支持的态度。
8. 自我——软件工程师应当参与终生职业实践的学习,并促进合乎道德的职业实践方法。

网络资源 关于 ACM/IEEE 的职业道德规范的完整讨论可在 seer.etsu.edu/codes/default.shtm 找到。

虽然八项原则中的每一项都同样重要,但最重要的一个主题是:软件工程师应该以公众的利益为目标。从个人的角度上,软件工程师应遵守以下规定:

- 决不将数据据为己有。
- 决不散布或出售在软件项目中工作时所获得的私有信息。
- 决不恶意毁坏或修改别人的程序、文件或数据。

- 决不侵犯个人、小组或组织的隐私。
- 决不闯入一个系统胡闹或牟取利益。
- 决不制造或传播计算机病毒。
- 决不使用计算技术去助长偏见或制造麻烦。

在过去的十年中，许多软件企业试图说服当权者批准保护条例 [SEE03]：(1) 允许公司在不公开已知缺陷的情况下发布软件；(2) 对于由这些已知缺陷所引起的任何损害，免除开发者的赔偿责任；(3) 没有得到原始开发者的允许，禁止其他人公开缺陷；(4) 允许将“自助”软件结合到一个产品中——这样能使产品丧失操作能力（通过远程命令）；(5) 如果第三方使软件丧失能力，免除使用“自助”能力的软件开发者的赔偿责任。

866

与所有的立法一样，争论的问题通常集中在政策方面，而不是技术方面。然而，很多人（包括我们大家）感到：如果没有合理地起草保护条例，而只是间接地免除软件工程师生产高质量软件的责任，那么保护条例将会与软件工程道德公约相冲突。

39.6 写在最后

自本书第1版的编写，至今已近35年。我仍然能够回忆起自己作为一位年轻的教授，坐在桌子旁为一本书撰写手稿的情形，当时这本书的主题几乎没有人关心，甚至很少有人理解。我还记得出版商的拒绝信，他（礼貌但坚定地）认为“软件工程”方面的书绝对不会有市场。幸运的是，McGraw-Hill 出版社决定尝试一下^①，其余的如他们所说的那样已经成为了历史。

在过去的30年中，本书发生了引人注目的变化——在范围、规模、风格和内容等方面。如软件工程一样，经过这么多年，它已经长大并且（我希望）成熟起来。

计算机软件开发工程方法目前仍然是传统的知识，虽然在“合适的范型”、敏捷的重要性、自动化程度以及最有效的方法等方面还存在争论，但软件工程的基本原则现在已在产业界得到普遍接受。然而，为什么直到最近我们才看见它们被广泛采用呢？

我认为答案是由于技术转变和伴随而来的文化变化的困难，即使我们大多数人认识到了软件需要工程学科；我们仍需要与过去的习惯作斗争，并且要面对一些容易重复过去所犯错误的新的应用领域（以及在这些领域工作的开发者）。为了使转变更容易，我们需要很多东西——一个灵活的、可适应的、明智的软件过程，更有效的方法，更强大的工具，实践者更好地接受和管理者的支持，以及大量的教育。

你不可能同意本书中描述的每种方法。某些技术和观点是相互矛盾的，为了在不同的软件开发环境中更好地发挥作用，必须对书中的某些部分进行调整。然而，我真诚地希望本书已经描述了我们面临的问题，展示了软件工程概念的优势并提供了方法和工具的框架。

当我们更深入地走进21世纪之时，软件依然是世界上最重要的产品和最重要的产业。它的影响和重要性已经经历了一段很长的路。然而，新一代的软件开发者必须迎接很多前一代人面临过的相同挑战。让我们期待迎接挑战的人——软件工程师——有更多的智慧去开发改善人类生活环境的系统。

867
868

① 实际上，这要归功于 Peter Freeman 和 Eric Munson，他们使 McGraw-Hill 相信，这是值得一试的。销售超过了100万册，公平地说，他们做了一个好决定。

索引

索引中的页码为英文原书页码,与书中页边标注的页码一致。

A

- Abstraction(抽象), 108, 232
- Access control(访问控制), 637
- Accessibility(可访问性), 336
- Action(动作), 16
- Activity diagram(活动图), 155, 180, 303, 869
- Actors(参与者), 149
- Aggregation objects(聚合对象), 633
- Aggregation(聚合), 873
- Agile alliance(敏捷联盟), 66
- Agile development(敏捷开发), 71, 见 agile process
- politics of(战略), 71
- Agile manifesto(敏捷宣言), 66
- Agile modeling principles(敏捷建模原则), 81
- Agile process(敏捷过程), 70
- Agile teams(敏捷团队), 691
- Agility(敏捷)
 - cost of change(变更成本), 69
 - definition of(定义), 68
 - principles(原则), 70
- Alpha test(α 测试), 485
- Analysis(分析)
 - defect(缺陷), 450
 - display content(显示内容), 331
 - rule of thumb(经验原则), 169
 - tasks(任务), 326
 - threat(威胁), 585
 - user(用户), 325
 - user interface(用户界面), 322, 325
 - work environment(工作环境), 331
- Analysis classes(分析类)
 - characteristics of(特征), 187
 - identifying(标识), 185
 - state diagrams(状态图), 204
- Analysis model(分析模型)
 - building(建立), 154
 - elements of(的元素), 154, 172
- Analysis packages(分析包), 199
- Annual loss expectancy(ALE, 年度预计损失), 594
- Application domains(应用领域), 6
- Application software(应用软件), 7
- Appraisal costs(评估成本), 422
- Archetype(原型), 269
- Architectural agility(体系结构灵活性), 280
- Architectural components(体系结构构件), 270
- Architectural context diagram(ACD, 体系结构环境图), 267
- Architectural decisions(体系结构决策), 246
- Architectural description language(ADL, 体系结构描述语言), 233, 276
- Architectural descriptions(体系结构描述), 255
- Architectural design metrics(体系结构设计度量), 663
- Architectural design tools(体系结构设计工具), 273
- Architecture(体系结构), 118, 253
 - alternatives(可选择的), 274
 - call and return(调用和返回), 260
 - conformance checking(一致性检查), 279
 - data centered(以数据为中心), 258
 - data flow(数据流), 259
 - definition of(的定义), 253
 - description template(描述模板), 257
 - genres(类型), 257
 - importance of(重要性), 254
 - mismatch(失调), 311
 - patterns(模式), 278
 - service-oriented(面向服务的), 266
 - structural properties(结构特性), 233
 - styles(风格), 258

tradeoffs (权衡), 275

WebApp Design (WebApp 设计), 381

Arguments for metrics (度量的观点), 720

Artificial intelligence software (人工智能软件), 7

Aspect-oriented software development (面向方面的软件开发), 54

Aspects (方面), 55, 237

Assessment, risk (评估, 风险), 777

Asset (资产), 585

Asset value (资产值), 591

Associations (关联), 198, 872

Assurance security (保证安全), 592

Attack (攻击), 585

Attack pattern (攻击模式), 589

Attack surface (攻击表层), 596

Attributes (属性), 188, 893

 metrics (度量), 654, 657

 quality (质量), 455

Audit trails (审核跟踪), 632

Audits, software quality (审核, 软件质量), 450

Automated debugging (自动调试), 491

Automated estimation techniques (自动估算技术), 743

Automated testing (自动测试), 571

Availability measures (可用性测量), 460

Avoiding management problems (避免管理问题), 697

B

Backlog (待定项), 79

Backtracking, debugging (回溯法, 调试), 491

Backward impact management (反向影响管理), 639

Baselines (基线), 626, 720

Basis path testing (基本路径测试), 500

Behavior models (行为模型), 203

Behavioral testing (行为测试), 509

Beta test (β 测试), 485

Big-bang integration testing (“一步到位式”集成测试), 476

Black box (黑盒), 604

 specification (规格说明), 605

 testing (测试), 509

Bootstrap, SPI framework (Bootstrap, 软件过程改进框架), 833

Bottom-up integration (自底向上集成), 477

Boundary classes (边界类), 892

Boundary value analysis (边界值分析), 512

Box structure specification (盒结构规格说明), 604

Brute force, debugging (蛮干, 调试), 491

Bugs (隐错), 432, 见 Faults, Defects

Building blocks (构造块), 848

Business classes (业务类), 897

Business process reengineering (BPR, 业务过程再工程), 799

 tools (工具), 801

Business risk (业务风险), 779

C

Casual meeting (临时会议), 439

Cause elimination (原因排除), 491

Certification (认证), 612

 model (模型), 612

 teams (团队), 611

 testing (测试), 489

 testing for MobileApps (测试移动 App), 570

Change control (变更控制), 626

 process (过程), 636

Change management (变更管理), 451, 631

Changes, types of (变更, 类型), 8

Checklist (检查单)

 MobileApp (移动 App), 570

 review (评审), 439

 risk item (风险项), 781

Chief programmer team (主程序员团队), 92

CK metrics suite (CK 度量集), 667

Class-oriented metrics (面向类的度量), 667

Class-Responsibility-Collaborator (CRC, 类-职责-协作者), 192

 cards (卡片), 74

Classes (类), 892, 见 object

 aggregate (聚合), 196

 analysis (分析), 185

 attributes (属性), 188

 boundary (边界), 892

- business (业务), 897
- characteristics (特征), 897
- cohesion (内聚), 898
- controller (控制器), 894
- coupling (耦合), 898
- design (设计), 239
- diagrams of (图), 156, 191, 870
- operations (操作), 189, 见 Methods
- persistent (持久), 897
- process (过程), 897
- testing (测试), 481, 528
- user interface (用户界面), 897
- Classic life cycle (经典生命周期), 42
- Cleanroom (净室)
 - design (设计), 607
 - process model (过程模型), 603
 - testing (测试), 610, 见 Statistical use testing
- Clear box (清晰盒), 605
 - Specification (规格说明), 607
- Closed paradigm team (封闭式范型团队), 689
- Cloud computing (云计算), 10, 97, 405
- Cluster testing (簇测试), 482, 529
- CMM (能力成熟度模型), 38
- CMMI (能力成熟度模型集成), 38, 828
- COCOMO II Model (COCOMO II 模型), 744
- Code quality (代码质量), 454
- Code restructuring (代码重构), 809
- Code reviews (代码评审), 433
- Coding principles (编码原则), 122
- Cohesion (内聚), 296, 898
- Collaboration (协作), 140, 195
 - diagram (图), 880
 - tools (工具), 98
- Collaboration development (协同开发), 852
- Communicaton activity (沟通活动), 17
 - principles of (原则), 110
 - diagram (图), 880
 - team (团队), 692
 - work tasks (工作任务), 696
- Compatibility tests (兼容性测试), 554
- Completion (完成), 484
- Complexity (复杂性)
 - managemnet of (管理), 845
 - elements (元素), 626
- Component model (构件模型), 612
- Component-based development (基于构件的开发), 53, 308
- Component-based software engineering (CBSE, 基于构件的软件工程), 308
- Component-based standards (基于构件的标准), 291
- Component-level metrics (构件级度量), 671
- Component-level testing (构件级测试), 555
- Components (构件)
 - adaptation (适应性), 310
 - class-based (基于类的), 291
 - classification (分类), 312
 - composition (组合 / 组装), 310
 - definition of (定义), 286
 - design guidelines (设计准则), 295
 - design of (设计), 290, 299
 - elaboration (细化), 287
 - object-oriented view (面向对象观点), 286
 - process-related view (过程相关的观点), 291
 - retrieving (检索), 312
 - traditional view (传统观点), 288
 - WebApp Design (WebApp 设计), 387
- Composition (组合), 873
- Concurrent models (并发模型), 49
- Condition testing (条件测试), 507
- Configuration audit (配置审核), 639
- Configuration management (配置管理), 626
- Configuration models (配置模型), 220
- Configuration objects (配置对象), 642
- Configuration review (配置评审), 484
- Configuration testing (配置测试), 558
- Connectivity testing (连接性测试), 483
- Consistency (一致性测试), 526
- Construction activity (构建活动), 17
 - principles (原则), 121
- Content (内容)
 - model (模型), 216
 - objects (对象), 379
 - management (管理), 643
 - repository (中心存储库), 630
 - testing (测试), 545
 - testing tools (测试工具), 547, 645
- Context ((项目) 环境), 694

Context aware apps (相关环境感知 App), 399
 Contingency planning (应急计划), 789
 Continues process improvement (持续过程改进), 449
 Control structure testing (控制结构测试), 507
 Controller class (控制器类), 894
 Conventional software test strategies (传统软件测试策略), 475
 Coordination and Communication (协调与沟通), 692
 Coordination, team (协调, 团队), 692
 Correctness checks (正确性检查)
 OOA model (面向对象分析模型), 525
 security (安全性), 591
 Correctness proof (正确性证明), 609
 Correctness verification (正确性验证), 608, 见
 Design verification
 Cost estimation tools (成本估算工具), 748
 Cost-effectiveness (成本效益), 436
 Coupling (耦合), 298
 class (类), 898
 CRC model review (类-职责-协作者模型评审), 527
 Critical practices (关键实践), 699
 Cross-cutting concerns (横切关注点), 54, 237, 589
 Customer voice table (客户意见表), 146
 Customers (客户), 111
 Myths about (有关神话), 24
 Cyclomatic complexity (环路复杂性), 503

D

Data design (数据设计), 244
 Data flow modeling (数据流建模), 511
 Data flow testing (数据流测试), 507
 Data restructuring (数据重构), 810
 Data tree (数据树), 217
 Database testing (数据库测试), 547
 Debugging (调试), 488
 automated (自动的), 492
 psychological considerations (心理因素/考虑), 491
 strategies (策略), 491

tactics (战术), 491
 Decision tree (决策树), 749
 Decomposition (分解),
 problem (问题), 694
 Decomposition techniques (分解技术)
 estimation (估算), 734
 Defect amplification (缺陷放大), 433
 Defect analysis (缺陷分析), 450
 Defect removal efficiency (DRE, 缺陷排除效率), 718
 Defects (缺陷), 432, 见 bug, faults
 Dependability (可靠性), 591
 Dependencies (依赖), 198, 872
 inversion (反向), 241
 management (管理), 628
 tracking (跟踪), 631
 Deployment (部署)
 activity (活动), 17, 125
 diagram (图), 874
 testing (测试), 487
 Deployment-level design (部署级设计), 248
 Depth-first integration (深度优先集成), 476
 Design (设计)
 aesthetic (美学), 377
 architectural (体系结构的), 267
 architectural elements (体系结构元素), 244
 classes (类), 239
 cleanroom (净室), 607
 component level (构件级), 247, 290, 299
 concepts (概念), 231
 content (内容), 379
 data (数据), 244
 deployment level (部署级), 248
 evolution of (演化), 230
 formal (形式化的), 603
 granularity (粒度), 365
 interfaces (界面), 245, 317
 metrics (度量), 663
 model (模型), 226
 navigation (导航), 285
 object-oriented concepts (面向对象概念), 238
 pattern-based (基于模式的), 347, 354
 postmodern (后现代), 854
 process (过程), 228

quality guidelines (质量准则), 228, 454
 refactoring (重构), 74
 refinement (细化), 237, 608
 reuse (复用), 312
 scenario-based design (基于场景的设计), 532
 task set (任务集), 231
 technical reviews (技术评审), 229
 test case (测试用例), 242, 515
 traditional components (传统构件), 307
 verification (验证), 608
 WebApps (WebApp), 371
 Desk check (桌面检查), 439
 Device compatibility testing (设备兼容性测试), 483
 Diagram (图)
 activity (活动), 869
 class (类), 870
 collaboration (协作), 880
 communication (沟通, 通信), 880
 deploiment (部署), 874
 sequence (顺序), 876
 state (状态), 884
 use-case (用例), 875
 Direct measures (直接测量), 708
 Document (文档)
 restructuring (重构), 804
 testing (测试), 517
 Domain analysis (域分析), 170
 Domain engineering (领域工程), 308
 Drivers (驱动程序), 469
 Dynamic Systems Development Method (DSDM, 动态系统开发方法), 79

E

Effort estimation tools (工作量估算工具), 748
 Elaboration (细化), 135, 237
 component (构件), 287
 task (任务), 327
 Elicitation (引导, 导出, 获取), 134
 agile (敏捷), 148
 work products (工作产品), 147
 Embedded Software (嵌入式软件), 7
 Empirical models (经验模型)

estimation (估算), 743
 Encapsulation (封装), 892
 End-users (最终用户), 111
 Engineering (工程)
 forward (正向), 811
 reverse (逆向), 805
 security (安全), 588
 Engineering/scientific software (工程 / 科学软件), 7
 Environmental resources (环境资源), 732
 Equivalence partitioning (等价类划分), 511
 Error correction (错误纠正), 492
 Error density (错误密度), 435
 Errors (错误), 432
 cost of (的成本), 423
 WebApp environment (WebApp 环境), 542
 Estimation (估算)
 agile development (敏捷开发), 746
 concepts (概念), 750
 decomposition techniques (分解技术), 734
 empirical models (经验模型), 743
 FP-based (基于功能点的), 738
 object-oriented projects (面向对象的项目), 746
 problem-based (基于问题的), 735
 process-based (基于过程的), 739
 reconciliation (调和), 742
 risk (风险), 782, 见 Risk projection
 software (软件), 727
 use case (用例), 740
 WebApp, 747

Evaluations, post-mortem (产品完成后评估), 445
 Exceptions (异常处理), 178
 Exhaustive testing (穷举测试), 500
 Exposure, risk (显露度, 风险), 786
 Extreme Programming (XP, 极限编程), 72
 activities (活动), 72
 industrial (产业化的), 75
 testing (测试), 75
 team (团队), 94

F

Failure costs (失效成本), 422

Failure curves (失效曲线)
 hardware (硬件), 5
 software (软件), 6
 Fault (故障), 432, 见 Bug, Defects
 Fault-based testing (基于故障的测试), 531
 Feasibility analysis (可行性分析)
 MobileApp (移动 App), 571
 Finite state modeling (有穷状态建模), 511
 Fire-fighting (救火), 778
 Flow graph (流图), 500
 Formal design (形式化设计), 603
 Formal methods (形式化方法), 53, 602
 tools (工具), 614
 Formal specification language (形式化规格说明语言), 900
 Formal technical review (FTR, 正式技术评审), 432, 441
 Forward engineering (正向工程), 811
 Forward impact management (正向影响管理), 638
 Framework (框架), 17, 291, 351
 activity (活动), 16, 17, 32
 SPI (软件过程改进), 819
 Function point (FP, 功能点), 659, 710
 estimation (估算), 738
 Functional decomposition (功能分解), 694
 Functional independence (功能独立性), 236
 Functional model (功能模型), 218
 Functional testing (功能测试), 509

G

Gap analysis (差距分析), 823
 Generalization (泛化), 871
 Genres (类型), 257
 Gesture testing (手语测试), 575
 Goal/question/metric (GQM, 目标 / 问题 / 度量), 656
 Graph matrices (图矩阵), 506
 Graph-based testing (基于图的测试), 509
 Graphic design (图形设计), 378

H

Hazards (灾难), 463, 790

Help facilities testing (帮助设施测试), 516
 Historical data (历史数据), 659
 Horizontal refinement (横向细化), 613
 Human elements (人员因素), 626
 Human resources (人力资源), 731
 Hype cycle (波动周期), 843

I

Identification (标识), 633
 risk (风险), 780
 Impact management (影响管理), 638
 risk (风险), 785
 Impact of risk (风险的影响), 785
 Inception (起始), 133
 Increments (增量), 44
 Independent program paths (独立程序路径), 502
 Independent test group (ITG, 独立测试组), 475
 Indicator (指标), 655
 Informal reviews (非正式评审), 439
 Information (信息), 3
 objectives (目标), 694
 representing (表示), 862
 Information hiding (信息隐蔽), 235
 Inheritance (继承), 894
 Inspections (审查), 432, 437
 Integration testing (集成测试), 475
 object-oriented (面向对象), 529
 work products (工作产品), 480
 Integration (集成)
 bottom-up (自底向上), 481
 top-down (自顶向下), 473
 Interaction frames (交互框), 878
 Interclass test-case design (类间测试用例设计), 534
 Interface analysis (界面分析), 325
 Interface design (界面设计), 245, 317
 evaluation of (评估), 342
 golden rules (黄金规则), 318
 issues (问题), 335
 MobileApps (移动 App), 341, 399
 models (模型), 323
 patterns (模式), 362
 process (过程), 323

steps (步骤), 332
 WebApps (WebApp), 337, 376
 Internationalization (国际化), 336, 578
 Internet of things (物联网), 588
 Invariant (不变式), 616
 Inventory analysis (库存目录分析), 803
 ISO 9001:2000, 38
 ISO 9001:2008, 462

L

Language (语言)
 formal specification (形式化规格说明), 900
 object constraint (对象约束), 887
 semantics (语义), 901
 syntax (语法), 901
 Z-specification (Z 规格说明), 904
 Layout (布局), 378
 Legacy software (遗留软件), 7
 Liability (责任), 425
 Living models (活性模型), 120
 Load testing (负载测试), 562
 LOC-based metrics (基于代码行的度量), 712
 Loop testing (循环测试), 507
 Lorenz and Kidd OO metrics (Lorenz 和 Kidd 提出的面向对象的度量), 671

M

Maintainability (可维护性), 797
 Maintenance (维护), 108
 software (软件), 796
 Make-buy decision (自行开发或购买的决策), 748
 Management (管理)
 myths (神话), 23
 project (项目), 684
 risk (风险), 451, 777
 security (安全), 451
 spectrum (范围), 685
 Manual test integration tools (人工测试集成工具), 614
 Maturity level (成熟度等级), 831
 Maturity models (成熟度模型), 821
 Measurement (测量), 654, 708
 principles (准则), 656

Measures (测度), 654
 availability (可用性), 460
 direct (直接), 708
 reliability (可靠性), 460
 Meetings (会议)
 casual (临时的), 439
 review (评审), 441
 Melding problem and process (合并问题和过程), 695
 Messages (消息), 895
 Methods (方法), 16, 893, 见 Operations
 Metrics (度量), 654
 architectural design (体系结构设计), 663
 arguments (争论), 720
 attributes (属性), 654, 657
 baseline (基线), 720
 business goals (业务目标), 724
 collection (收集), 721
 small organizations (小型组织), 721
 program establishment (制定大纲), 722
 class-oriented (面向类的), 667
 component-level (构件级), 671
 design (设计), 663
 function-based (基于功能), 659, 710
 LOC-based (基于代码行), 712
 MobileApp design (移动 App 设计), 673
 object-oriented (面向对象), 666, 713
 private (私有的), 706
 process (过程), 704
 productivity (生产率), 712
 project (项目), 707
 public (公有的), 706
 quality (质量), 456
 requirements model (需求模型), 659
 reviews (评审), 534
 size-oriented (面向规模), 709
 software quality (软件质量), 716
 source code (源代码), 675
 specification quality (规格说明质量), 663
 testing (测试), 676
 use case (用例), 714
 user interface design (用户界面设计), 672
 WebApp, 714
 WebApp Design (WebApp 设计), 673

Middleware (中间件), 405
 MobileApps (mobile application)(移动 App), 9
 architectural design of (体系结构设计), 274
 best design practices (最佳设计实践), 401
 checklist (检查单), 570
 component-level design (构件级设计), 306
 context aware (感知范围), 399
 design (设计), 391
 design challenges (设计挑战), 392
 design mistakes (设计错误), 401
 design metrics (设计度量), 673
 development activities (开发活动), 395
 interface design (界面设计), 398
 patterns (模式), 366
 quality checklist (质量检查单), 397
 real-time testing (实时测试), 578
 requirements modeling (需求建模), 214
 SCM (软件配置管理), 640
 software engineering (软件工程), 407
 stress testing (压力测试), 573
 test matrix (测试矩阵), 572
 testing (测试), 483
 testing guidelines (测试准则), 568
 testing strategies (测试策略), 569
 testing tools (测试工具), 579
 tools (工具), 404
 usability testing (可用性测试), 575
 Miobility environments (可移动环境), 403
 Model-based testing (基于模型的测试), 514
 Model-driven software development (模型驱动的软件开发), 853
 Model-view-controller (MVC, 模型-视图-控制器结构), 384
 Model (模型)
 certification (验证), 612
 COCOMO II, 744
 component (构件), 612
 sampling (取样), 612
 security (安全性), 590
 modeling (建模), 17
 agile (敏捷), 80
 CRC (类-职责-协作者), 192
 data flow (数据流), 511
 finite state (有限状态), 511

principles (原则), 114
 scenario-based (基于场景), 173
 security (安全), 590
 threat (威胁), 594
 timing (定时), 511
 transaction (事务), 511

Models (模型)

behavioral (行为), 203
 design (设计), 243
 object-oriented (面向对象), 525
 requirements (需求), 167

Modularity (模块化), 234

Module (模块), 288, 见 Component

MOI leadership model (激励-组织-思想与创新领导模型), 688

Mongolian horde concept (蒙古游牧概念), 24

MOOD metrics suite (MOOD 度量集), 670

Multiple class testing (多类测试), 534

Multiplicity (重数), 872

Myths (神话), 23

customer (客户), 24
 management (管理), 23
 practitioner (实践者), 25

N

Navigation (导航)

semantic unit (语义单元), 387
 semantics (语义), 385
 syntax (语法), 387

Navigation modeling (导航建模), 220

Navigation testing (导航测试), 556

Negligence (疏忽), 425

Negotiation (谈判), 135, 159

O

object (对象), 892, 见 Class

Object constraint language (OCL, 对象约束语言), 887, 901

example of (实例), 903

notation (表示法), 902

Object-oriented (面向对象)

analysis (分析), 172, 184

- design metrics (设计度量), 666
 - integration testing (集成测试), 529
 - metrics (度量), 713
 - model consistency (模型一致性), 526
 - models (模型), 525
 - OOA model correctness (面向对象分析模型正确性), 525
 - OOD model correctness (面向对象设计模型正确性), 525
 - project estimation (项目估算), 746
 - software (软件), 481
 - test case design (测试用例设计), 530
 - testing strategies (测试策略), 528
 - unit testing (单元测试), 528
 - validation testing (确认测试), 529
 - OO metrics, Lorenz and Kidd (Lorenz 与 Kidd 提出的面向对象度量), 671
 - Open paradigm team (开放式范型团队), 689
 - Open source (开源), 848
 - Open world computing (普适计算), 846
 - Open-closed principle (开放-封闭原则), 292
 - Operations (操作), 189, 893, 见 Methods
 - Organizational paradigms (组织范型), 92
 - Orthogonal array testing (正交数组测试), 513
 - Outsourcing (外包), 750
- P**
- Pair programming (结对编程), 75, 440
 - Partitioning testing (划分测试), 533
 - Partitioning (划分), 694
 - Pattern organizing table (模式组织表), 358
 - Patterns (模式), 109
 - analysis (分析), 157, 207
 - architectural (体系结构), 263, 278, 359
 - component-level (构件级), 360
 - context (上下文), 348
 - describing (描述), 352
 - design (设计), 348
 - example of (实例), 209
 - interface design (界面设计), 362
 - kinds of (类型), 349
 - languages (语言), 353
 - MobileApps (移动 App), 366
 - process (过程), 35
 - repository (中心存储库), 353, 360
 - requirements modeling (需求建模), 207
 - template for (所用模板), 352
 - testing (测试), 519
 - WebApps (WebApp), 364
 - Peer reviews (同行评审), 432
 - People (人员), 687
 - People Capability Maturity Model (人员能力成熟度模型), 685, 832
 - People software process (PSP, 人员软件过程), 59, 833
 - Performance testing (性能测试), 487, 580
 - Persistent classes (持久类), 897
 - Phishing ((钓鱼式攻击) 网上行骗者), 587
 - Plan, RMMM (风险缓解, 监测管理计划), 790
 - Plan, SQA (软件质量保证计划)
 - Planning activity (策划活动), 17
 - principles (原则), 112
 - web testing (Web 测试), 543
 - Plug points (插入点), 351
 - Polymorphism (多态性), 896
 - Post-mortem evaluations (PME, 产品完成后评估), 445
 - Postconditions (后置条件), 616
 - Practice (software engineering) (实践 (软件工程))
 - core principles (核心原则), 108
 - essence of (精髓), 19
 - Pre-condition (前置条件), 178, 616
 - Predictable costs (可预测风险), 779
 - Prevention costs (预防成本), 422
 - Priority points (优先点), 140
 - Privacy (保密), 586
 - cloud computing (云计算), 587
 - social media (社会媒体), 587
 - Private metrics (私有度量), 706
 - Proactive risk strategies (主动风险策略), 778
 - Problem decomposition (问题分解), 694
 - Problem elaboration (问题细化), 694
 - solving (问题求解), 19
 - Problem-based estimation (基于问题的估算), 735
 - Process (过程), 16, 686
 - adaptation (适应), 18
 - agile (敏捷), 69

- assessment (评估), 37
 - classes (类), 897
 - components of (构件), 16
 - debugging (调试), 490
 - decomposition (分解), 694
 - elements (元素), 626
 - extreme programming (极限编程), 72
 - flow (流), 31, 33
 - framework (框架), 17, 32
 - generic (通用), 31
 - iterative (迭代), 31
 - immaturity (不成熟), 822
 - pattern template (模式模板), 35
 - patterns (模式), 35, 37
 - principles (原则), 106
 - relationship to product (与产品的关系), 62
 - SCM (软件配置管理), 632
 - Process improvement (过程改进), 37, 818
 - approaches (方法), 819
 - continuous (持续), 449
 - education (教育), 825
 - evaluation (评价), 827
 - gap analysis (差距分析), 823
 - installation (设置), 826
 - justification (合理性判定), 825
 - other frameworks (其他框架), 832
 - ROI (投资回报率), 834
 - trends (趋势), 835
 - Process maturity (过程成熟度), 821
 - Process metrics (过程度量), 704
 - Process modeling, tools (过程建模, 工具), 62
 - Process models (过程模型), 40
 - cleanroom (净室), 603
 - concurrent (并发 / 并行), 49, 51
 - evolutionary (演化式), 45
 - incremental (增量式), 43
 - prescriptive (惯例), 41
 - risk driven (风险驱动), 48
 - software reengineering (软件再工程), 803
 - specialized (专用), 52
 - Process technology (过程技术), 61
 - Process-based estimation (基于过程的估算), 739
 - Producer (生产者), 441
 - Product (产品), 686, 693
 - Product metrics (产品度量), 653
 - tools (工具), 678
 - Product-line software (产品线软件), 7, 11
 - Product-specific risk (产品特定风险), 780
 - Productivity measures (生产率测度), 712
 - Productivity metrics (生产率度量), 712
 - Project (项目), 686, 693
 - Project complexity (项目复杂度), 728
 - Project database (项目数据库), 627
 - Project management (项目管理), 684
 - tools (工具), 699
 - Project metrics (项目度量), 707
 - Project planning (项目计划), 729
 - process (过程), 729
 - task set (任务集), 730
 - Project risk (项目风险), 779
 - Project size (项目规模), 729
 - Project velocity (项目速度), 73
 - Projects, getting started (项目, 启动), 26
 - Projection, risk (预测, 风险), 782, 见 Risk estimation
 - Prototyping (原型开发), 45
 - problem with (问题), 46
 - Public metrics (公有度量), 706
- Q**
- Quality (质量)
 - attributes (属性), 455
 - “good enough” (“足够好”), 421
 - code (代码), 455
 - concepts (概念), 412
 - cost of (成本), 422
 - definition of (定义), 413
 - design (设计), 454
 - Garvin’s dimensions (Garvin 质量维度), 415
 - ISO 9126, 418
 - management actions (管理活动), 426
 - McCall’s factors (McCall 质量因素), 416
 - methods for achieving (获取方法), 427
 - metrics (度量), 455
 - MobileApps (移动 App), 397
 - quantitative view (量化观点), 373
 - requirements (需求), 454

- requirements tree (需求树), 420
- security (安全), 425
- WebApp Design (WebApp 设计), 372
- Quality assurance (质量保证), 428
 - statistical (统计), 456
- Quality control (质量控制), 427
- Quality function deployment (QFD, 质量功能部署), 146
- Quality management (质量管理), 449
 - resources (资源), 452
- R**
- Random paradigm team (随机范型团队), 689
- Random testing (随机测试), 532
- Rapid cycle testing (快速周期测试), 473
- Reactive risk strategies (被动风险策略), 778
- Real-time testing (实时测试)
 - MobileApps (移动 App), 578
 - System (系统), 517
- Reconciliation, estimation (协调, 估算), 742
 - LOC and FP metrics (代码行度量及功能点度量), 711
- Record keeping (记录保持), 442
- Recorder (记录员), 442
- Recovery testing (恢复测试), 486
- Reengineering (再工程)
 - business process (业务过程), 799
 - economics (经济学), 813
 - software (软件), 802
- Refactoring (重构), 74, 238, 301
- Refinement (细化), 237, 见 Elaboration
- Regression testing (回归测试), 478
- Reliability (可靠性)
 - measures (测度), 459
 - software (软件), 459
- Repository (中心存储库), 630
 - content (内容), 630
 - design patterns (设计模式), 360
 - hypermedia (超媒体), 365
 - SCM (软件配置管理), 630
- Requirements (需求)
 - emergent (意外), 846
 - negotiating (谈判), 159
 - nonfunctional (非功能), 141
 - security (安全性), 585
 - understanding (理解), 131
 - validation (确认), 161
- Requirements elicitation (需求获取)
 - security (安全), 589
- Requirements engineering (需求工程), 132, 852
 - agile (敏捷), 158
 - common mistakes (共性错误), 162
 - first questions (第一个问题), 140
 - goal-oriented (面向目标), 134
 - tools (工具), 138
- Requirements gathering (需求收集)
 - collaborative (协作), 143
- Requirements modeling (需求建模)
 - approaches to (方法), 171
 - Objectives (目标), 168
 - principles (原则), 116
 - WebApps (WebApp), 213
 - MobileApps (移动 App), 213
- Requirements models (需求模型), 114, 见 Analysis models
 - metrics (度量), 659
 - types of (类型), 167
- Requirements quality (需求质量), 454
- Requirements specification (需求规格说明)
 - template for (模板), 136
- Requirements tasks (需求任务)
 - elaboration (细化), 135
 - elicitation (获取), 134, 142
 - inception (起始), 133
 - negotiation (协商), 135
 - specification (规格说明), 135
 - validation (确认), 136
- Requirements tracing (需求跟踪), 631
- Resources (资源), 731
 - environmental (环境), 732
 - human (人员), 731
 - quality management (质量管理), 452
 - reusable software (复用软件), 732
- Responsibilities (职责), 193
- Restructuring (重构), 809
 - code (代码), 809
 - data (数据), 810

- document (文档), 804
 - Reusable software resources (可复用软件资源), 732
 - Reuse (复用), 312
 - Reverse engineering (逆向工程), 805
 - data (数据), 807
 - processing (处理), 807
 - tools (工具), 809
 - user interfaces (用户界面), 808
 - Review (评审)
 - checklist (检查单), 439
 - guidelines (准则), 442
 - issues list (问题清单), 442
 - leader (主持), 441
 - meeting (会议), 441
 - metrics (度量), 435
 - reporting (报告), 442
 - Review information sheet (RIS, 评审信息表单), 790
 - Reviews (评审)
 - architectural (体系结构), 277
 - code (代码), 433
 - configuration (配置), 488
 - informal (非正式), 439
 - peer (同行), 432
 - sample-driven (模板驱动), 444
 - software quality (软件质量), 450
 - technical (技术), 441
 - Risk (风险)
 - assessment (评估), 777
 - categories (类型), 779
 - components (因素), 782
 - drivers (驱动因子), 782
 - estimation (估计), 782, 见 risk projection
 - exposure (显露度), 786
 - identification (识别), 780
 - impact (影响), 785
 - information sheet (信息表单), 790
 - item checklist (条目检查表), 781
 - patterns (模式), 781
 - projection (预测), 782, 见 risk estimation
 - refinement (细化), 787
 - Risk management (风险管理), 777
 - principles (原则), 780
 - tools (工具), 791
 - SPI (软件过程改进), 827
 - Risk Mitigation, Monitoring, and Management (RMMM, 风险缓解、监测和管理), 788, 790
 - Risk strategies (风险策略)
 - proactive (主动), 778
 - reactive (被动), 778
 - Risk table (风险表), 783
 - Risks (风险), 424
 - business (商业), 779
 - known (已知), 779
 - predictable (可预测), 779
 - produce-specific (产品特定), 780
 - project (项目), 779
 - technical (技术), 779
 - unpredictable (不可预测), 779
- S
- SafeHome (住宅保安系统), 143, 150
 - activity diagram (活动图), 220
 - analysis patterns (分析模式), 209
 - applying patterns (应用模式), 362
 - architectural assessment (体系结构评估), 276
 - architectural decisions (体系结构的设计决策), 265
 - architecture context diagram (体系结构环境图), 269
 - behavioral modeling (行为建模), 157
 - choosing an architecture (选择体系结构), 262
 - class models (类模型), 190
 - cohesion in action (内聚性的应用), 297
 - communication mistakes (沟通失误), 111
 - conclusion (结论), 862
 - considering agile process (考虑敏捷过程), 76
 - coupling in action (耦合的应用), 298
 - CRC models (CRC 模型), 197
 - data tree (数据树), 217
 - design classes (设计类), 241
 - design concepts (设计概念), 239
 - design patterns (设计模式), 356
 - design vs. coding (设计与编码比较), 227
 - domain analysis (域分析), 171

- getting started (起动, 开始), 26
- grammatical parse (语法分析), 186
- graphic design (图示设计), 377
- instantiations of (实例), 272
- interface design review (界面设计评审), 340
- interface golden rules (界面的黄金规则), 320
- MobileApp requirements (移动 App 需求), 396
- negotiation (协商), 160
- open closed principle (开闭原则), 293
- preliminary user scenario (初始用户场景), 174
- quality issues (质量问题), 424
- requirements gathering (需求收集), 145
- screen layout (屏幕布局), 334
- selecting a model (选定模型), 47, 50
- sequence diagram (顺序图), 206
- state diagram (状态图), 205
- swimlane diagram (泳道图), 181
- team structure (团队结构), 93
- use case diagram (用例图), 153
- use case for UI design (用户界面设计的用例), 327
- use case template (用例模板), 178
- user scenarios (用户场景), 147
- Safety (安全), 451, 790
 - software (软件), 460
- Sample-driven reviews (样本驱动评审), 444
- Sampling model (取样模型), 612
- Scenario-based testing (基于场景的测试), 532
- Scope (范围), 113, 134
 - software (软件), 694
- Scrum, 78
 - meetings (会议), 79
- Security and privacy (安全性及私密性), 586
- Security assurance (安全性保证), 592
- Security assurance case (安全性保证用例), 592, 见 Trustworthiness
- Security engineering (安全性工程), 588
 - tools (工具), 598
 - user-centered (以用户为中心), 589
- Security (安全性)
 - case (实例, 横切), 591
 - management (管理), 451
 - correctness checks (正确性检查), 591
 - model (模型), 590
 - Objectives (目的), 590
 - requirements elicitation (需求获取), 585, 589
 - quality aspects (质量状况), 425
- Self-adaptive systems (自适应系统), 158
- Self-organization teams (自组织团队), 692
- Semantics, language (语义, 语言), 901
- Separation of concerns (关注点分离), 108, 234
- Sequence diagram (顺序图), 205, 876
- Service-oriented methods (面向服务的方法), 148
- Services (服务), 893
- Six Sigma (六西格玛), 458
- Size-oriented metrics (面向规模的度量), 709
- Smoke testing (冒烟测试), 479
- Social media (社交媒体), 95
- Social media and privacy (社交媒体与保密性), 587
- Software architecture (软件体系结构), 见 Architecture
- Software component (软件构件), 见 Component
- Software configuration audit (软件配置审核), 639
- Software configuration items (SCI, 软件配置项), 628
- Software configuration management (SCM, 软件配置管理), 623
 - MobileApps (移动 App), 640
 - process (过程), 632
 - repository (中心存储库), 630
 - standards (标准), 649
 - tools (工具), 640
 - WebApps (WebApp), 640
- Software defects, cost impact (软件缺陷, 成本影响), 432
- Software design, (软件设计), 见 Design
- Software development, myths (软件开发, 神话), 23
- Software engineer (软件工程师)
 - characteristic of (特点), 88
 - roles (作用), 89
- Software engineering (软件工程)
 - cloud-based (基于云), 97
 - component-based (基于构件), 308
 - core concepts (核心概念), 106
 - definition of (定义), 15
 - design (设计), 225

- ethics (道德规范), 865
- grand challenge (巨大挑战), 851
- impact of social media (社会媒体的影响), 95
- layers of (层次), 16
- long view (远景), 864
- practice (实践), 19
- principles (原则), 21, 104
- psychology (心理学), 89
- technology directions (技术方向), 849
- trends (趋势), 839
- video games (视频游戏), 1
- work practices (工作实践), 126
- Software equation (软件方程), 744
- Software estimation (软件估算), 727
- Software maintenance (软件维护), 796, 见 Maintenance
- Software Process (软件过程), 16
- Software process improvement (SPI, 软件过程改进), 见 Process improvement
- Software quality assurance (SQA, 软件质量保证), 448
 - elements of (元素), 450
- Software quality (软件质量), 见 Quality
 - audits (审核), 450
 - metrics (度量), 716
 - reviews (评审), 450
 - standards (标准), 450
- Software reengineering (软件再工程), 802
 - process model (过程模型), 803
- Software reliability (软件可靠性), 459
- Software restructuring tools (软件重构工具), 810
- Software safety (软件安全), 460
- Software scope (软件范围), 694, 730, 见 Scope
- Software sizing (软件规模估算), 734
- Software teams (软件团队), 689, 见 Teams
- Software (软件)
 - application domains (应用领域), 6
 - as capital (作为资产), 30
 - definition of (定义), 4
 - “good enough” (“足够好”), 421
 - importance of (重要性), 2, 861
 - key questions (关键问题), 4
 - nature of (性质), 3
 - object-oriented (面向对象), 487
 - realities (事实), 14
- Source code metrics (源代码度量), 675
- Specification (规格说明), 135
 - black-box (黑盒), 605
 - box structure (盒结构), 604
 - clear-box (清晰盒), 607
 - quality metrics (质量度量), 663
 - state-box (状态盒), 606
- SPICE (软件过程改进与能力确定), 38, 833
- Spike solution (Spike 解决方案), 74
- Spiral model (螺旋模型), 47
- Sprints (冲刺), 79
- SQA group (软件质量保证组), 450
- SQA plan (软件质量保证计划), 454
- Stakeholder (利益相关者), 687
 - definition of (定义), 17
 - identifying (识别), 139
- Standards (标准)
 - ISO 9001: 2008, 462
 - SCM (软件配置管理), 649
 - software quality (软件质量), 450
- State box (状态盒), 604
- State diagram (状态图), 884
- State representations (状态表示), 204
- State-box specification (状态盒规格说明), 606
- Statechart (状态图), 304
- Statistical quality assurance (统计质量保证), 456
- Statistical software process improvement (统计软件过程改进, SSPI), 707
- Statistical use testing (统计使用测试), 604
- Status reporting, configuration (状态报告, 配置), 639
- Stereotype (构造型), 871
- Strategy (策略)
 - debugging (调试), 491
 - testing (测试), 471
- Stress testing (压力测试), 487, 562
 - MobileApp (移动 App), 573
- Structure chart (结构图), 289
- Structured analysis (结构化分析), 171
- Stub (桩模块), 486
- Subclass (子类), 893
- Superclass (超类), 893
- Supportability (可支持性), 798

Surface, attack (表面, 攻击), 596
 Survivability (生命力, 存活性), 591
 Swimlane diagram (泳道图), 181, 330
 Synchronization control (同步控制), 637
 Synchronous paradigm team (同步范型团队), 690
 Syntax (句法), 901
 System classes (系统类), 897
 System of forces (影响因素), 348
 System software (系统软件), 6
 System testing (系统测试), 486
 System vulnerability (系统脆弱性), 591

T

Task (任务), 16
 Task analysis (任务分析), 326
 Task set (任务集), 31
 identification of (标识), 34
 project planning (项目计划), 730
 Team leader (团队负责人), 688
 Team Software Process (TSP; 团队软件过程), 60, 833
 Teams (团队), 90
 agile (敏捷), 93, 691
 closed paradigm (封闭式范型), 689
 global (全球的), 99
 jelled (凝聚的), 90
 open paradigm (开放式范型), 689
 organizational paradigm (组织范型), 92
 random paradigm (随机式范型), 689
 self-organizing (自组织), 94, 692
 structures (结构), 92
 synchronous paradigm (同步式范型), 690
 talent mix (人才技能结合), 847
 toxicity (毒性), 91
 XP (极限编程), 94
 Technical review reference model (技术评审参考模型), 438
 Technical reviews (技术评审), 441
 Technical risk (技术风险), 779
 Technology evolution (技术演变), 840
 Test across borders (跨界测试, 578, 见 Internationalization)
 Test case derivation (测试用例导出), 504

Test case design (测试用例设计), 515
 Test (测试)
 alpha (α), 485
 beta (β), 477
 characteristics (特征), 498
 extraction tools (提取工具), 614
 generation tools (生成工具), 614
 object-oriented (面向对象), 530
 strategies for conventional software (传统软件的策略), 473
 Test matrix (测试矩阵), 572
 Test-driven development (测试驱动开发), 854
 Testability (可测试性), 497
 Testing (测试)
 automated (自动化), 571
 basis path (基本路径), 500
 behavioral (行为), 509
 black box (黑盒), 509
 certification (认证), 570
 class (类), 482, 528
 cleanroom (净室), 610, 见 Statistical use testing
 cluster (簇测试), 529, 472
 component-level (构件级), 555
 condition (条件), 507
 control structure (控制结构), 507
 criteria (标准), 472
 data flow (数据流), 507
 database (数据库), 547
 deployment (部署), 475
 documentation (文档), 517
 exhaustive (穷举), 500
 fault-based (基于故障), 531
 functional (功能), 509
 gesture (手势), 575
 graph-based (基于图), 509
 guidelines (准则), 568
 help facilities (帮助设施), 516
 integration (集成), 481
 loop (循环), 507
 methods (方法), 529, 676
 mobile-app (移动 App), 482
 model-based (基于模型), 514
 multiple class (多类), 534
 object-oriented (面向对象), 528

- orthogonal array (正交数组), 513
- partition (划分), 533
- patterns (模式), 519
- performance (性能), 486
- principles (原则), 123
- process (过程), 544
- Testing, random (测试, 随机), 532
- real-time system (实时系统), 517
- recovery (恢复), 486
- regression (回归), 486
- scenario-based (基于场景), 532
- security (安全), 479
- smoke (冒烟), 487
- statistical use (统计应用), 604
- strategies (策略), 528, 543, 569, 470
- stress (压力), 475
- system (系统), 483
- thread-based (基于线程), 476, 529
- tools (工具), 579
- unit (单元), 468
- use-based (基于使用), 529
- validation (验证), 468
- WebApp, 482
- white-box (白盒), 500
- Testing-in-the-wild (狂测), 483, 573
- Thread-based testing (基于线程的测试), 482, 529
- Threat (威胁)
 - analysis (分析), 585
 - likelihood (似真), 591
 - modeling (建模), 594
- TickIT, 834
- Timing modeling (时间建模), 511
- Tools (工具)
 - agile process (敏捷过程)
 - architectural description languages (体系结构描述语言), 277
 - architectural design (体系结构设计), 273
 - architectural conformance (体系结构一致性), 280
 - business process reengineering (BPR, 业务过程再工程), 801
 - CBSE (CBSE, 基于构件的软件工程), 313
 - collaboration (协作), 98
 - content management (内容管理), 645
 - cost estimation (成本估算), 748
 - effort and cost estimation (工作量与成本估算), 748
 - effort estimation (工作量估算), 748
 - formal methods (形式化方法), 614
 - manual test integration (手工测试集成), 614
 - MobileApp testing (移动 App 测试), 580
 - process modeling (过程建模), 62
 - product metrics (产品度量), 678
 - project and process metrics (项目与过程度量), 716
 - project management (项目管理), 699
 - requirements engineering (需求工程), 138
 - reverse engineering (逆向工程), 809
 - risk management (风险管理), 791
 - SCM (软件配置管理), 640
 - security engineering (安全工程), 598
 - Software restructuring (软件重构), 810
 - test extraction (测试抽取), 614
 - test generation (测试生成), 614
 - use cases (用例), 154
 - user interface development (用户界面开发), 337
 - user-interface testing (用户界面测试), 554
 - version control (版本控制), 635
 - web configuration testing (Web 配置测试), 559
 - web content testing (Web 内容测试), 547
 - web navigation tools (Web 导航测试), 557
 - web performance testing (Web 性能测试), 563
 - web security testing (Web 安全测试), 560
 - WebApp change management (WebApp 变更管理), 647
 - WebApp Metrics (WebApp 度量), 675
 - Top-down integration (自顶向下集成), 476
 - Toxicity, team (毒性, 团队), 690
 - Traceability (可追溯性), 118, 142
 - matrix (矩阵), 142, 628
 - Tracking (跟踪)
 - dependency (依赖性), 631
 - Transaction modeling (事务建模), 511
 - Trends (趋势)
 - process (过程), 849
 - tools (工具), 855
 - Trigger (启动), 178

Trust verification (信任验证), 591

Trustworthiness (可信性), 591

U

Umbrella activities (普适性活动), 17

UML Notation (统一建模语言表示法), 869

activity diagram (活动图), 155, 180, 303

analysis modeling (分析建模), 207

class diagram (类图), 156

deployment diagram (部署图), 248

interface representation (界面表示), 246

state diagram (状态图), 156

swimlane diagram (泳道图), 181

use-case diagram (用例图), 153

Uncertainty (不确定性), 779

Unified Process (统一过程), 55

agile (敏捷), 82

history of (历史), 56

phases (阶段), 56

workflow (工作流), 58

Unit testing (单元测试), 473

object-oriented (面向对象), 528

Usability (可用性), 322

testing (测试), 552, 575

Usage scenarios (使用场景), 146

Use-based testing (基于使用的测试), 529

Use case (用例), 875

estimation (估算), 740

Use case points (UCP, 用例点), 742

Use cases (用例), 146, 326

creating (生成), 173

developing (开发), 149

diagram (图), 153, 179, 875

estimation (估算), 740

events (事件), 203

formal (正式的), 177

metrics (度量), 714

question to be answered (要回答的问题), 150

refinement of (细化), 176

tools (工具), 154

User interface classes (用户界面类), 897

User interface design (用户界面设计), 见 Interface design

User interface design metrics (用户界面设计度量), 672

User interface testing (用户界面测试), 549

User stories (用户故事), 73

User-centered security engineering (以用户为中心的安全工程), 589

Use experience testing (用户体验测试), 483

V

V-model (V 模型), 43

Validation (确认), 136, 161, 483

Validation testing (确认测试), 483

checklist (检查单), 137

object-oriented (面向对象), 529

Value adjustment factors (VAF, 值调整因子), 660

Verification (验证), 470

correctness (正确性), 608, 见 design verification

design (设计) 608, 见 design verification

trust (信任), 591

Version control (版本控制), 634

tools (工具), 635

Versioning (版本管理), 631

Vertical refinement (纵向细化), 613

Viewpoints, multiple (视点, 多重), 139

Virtual keyboard input (虚拟键盘输入), 577

Voice input (语音输入), 576

Voice recognition (语音识别), 576

Volatility (易变性), 134

W

W⁵HH Principle (W⁵HH 原则), 698

Walkthroughs (走查), 432

Waterfall model (瀑布模型), 41

problems with (相关问题), 42

Web (网络)

configuration testing tools (配置测试工具), 559

navigation testing tools (导航测试工具), 557

performance testing tools (性能测试工具), 563

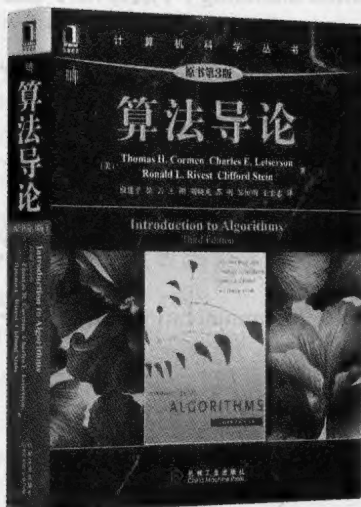
quality dimensions (质量维度), 541

security testing tools (安全测试工具), 560

test planning (测试计划), 543

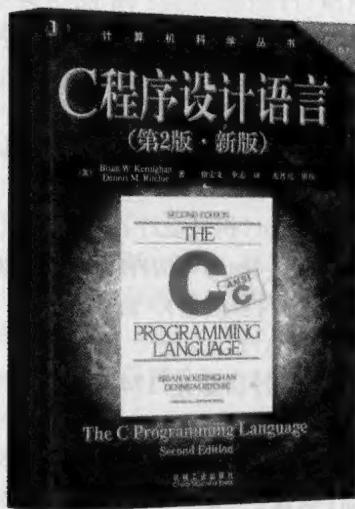
- testing process (测试过程), 544
 - testing strategy (测试策略), 543
 - user-interface (UI) testing tools (用户界面测试工具), 554
 - Web/mobile applications (Web/ 移动 App), 7
 - WebApps (Web applications)(WebApp), 9
 - architectural design of (体系结构设计), 273
 - architecture (体系结构), 381
 - change management tools (变更管理工具), 647
 - characteristics (特性), 379
 - component design (构件设计), 387
 - component-level design (构件级设计), 305
 - content model (内容模型), 216
 - design (设计), 371
 - design goals (设计目标), 374
 - design metrics (设计度量), 673
 - environment errors (环境错误), 542
 - estimation (估算), 747
 - metrics (度量), 714
 - metrics tools (度量工具), 675
 - SCM (软件配置管理), 640
 - design pyramid (设计金字塔), 375
 - functional model (功能模型), 218
 - interface design (界面设计), 376
 - managing change (管理变更), 647
 - navigation modeling (导航建模), 220
 - quality of (质量), 372
 - requirements modeling (需求建模), 213
 - testing (测试), 482
 - White-box testing (白盒测试), 500
 - Work practices (工作实践), 126
 - Work products, integration testing (工作产品, 集成测试), 483
 - Work tasks, communication (工作任务, 沟通), 696
 - Workflow analysis (工作流分析), 328
- Z**
- Z Specification language (Z 规格说明语言), 904
 - example of (实例), 906
 - notation (表示方法), 905

推荐阅读



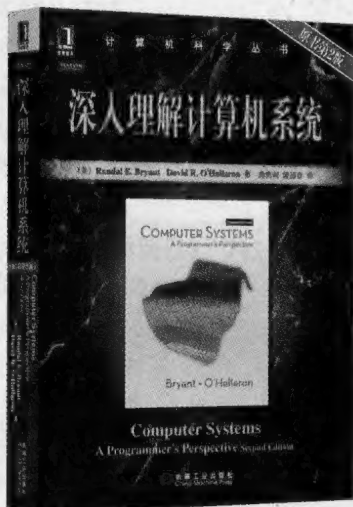
算法导论（原书第3版）

作者：Thomas H. Cormen 等著
ISBN: 978-7-111-40701-0 定价：128.00元



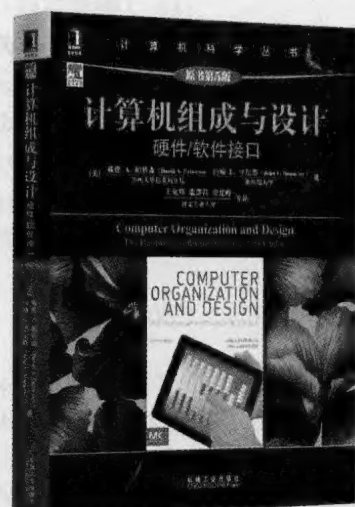
C程序设计语言（第2版·新版）

作者：Brian W. Kernighan 等著
ISBN: 978-7-111-12806-0 定价：30.00元



深入理解计算机系统（原书第2版）

作者：Randal E. Bryant 等著
ISBN: 978-7-111-32133-0 定价：99.00元



计算机组成与设计：硬件/软件接口（原书第5版）

作者：David Patterson, John Hennessy
ISBN: 978-7-111-50482-5 定价：99.00元

软件工程 实践者的研究方法 原书第8版

Software Engineering A Practitioner's Approach Eighth Edition

本书是软件工程领域的经典权威著作，自第1版出版至今，30多年来在软件工程界产生了巨大而深远的影响。第8版在继承之前版本风格与优势的基础上，不仅更新了全书内容，而且优化了篇章结构。本书共五个部分，涵盖软件过程、建模、质量管理、项目管理等主题，对概念、原则、方法和工具的介绍细致、清晰且实用。此外，书中提供了丰富的辅助阅读资源并配有网络资源（www.mhhe.com/pressman）。

本书特色

- 重点与更新。突出软件质量管理的相关内容，同时加强了软件过程部分。与时俱进的内容包括移动App开发、软件系统安全性和软件工程对人员的要求。
- 组织与结构。每章开篇给出“要点浏览”和“关键概念”，最后给出“小结”“习题与思考题”以及“扩展阅读与信息资源”，章中贯穿的辅助阅读信息包括SafeHome对话框、软件工具、引述、关键点、网络资源等。
- 教学与自学。新的篇章结构更便于教师针对不同课程选取素材，同时，全面的知识点和丰富的扩展资源不仅易于初学者掌握基础理论，而且可供有兴趣的读者进行深入研究。

作者简介

罗杰 S. 普莱斯曼（Roger S. Pressman）软件工程界国际知名的顾问和作家，作为工程师、经理人、教授、演讲家和企业家奋战在这一领域已余40年。现任一家咨询公司的总裁，致力于协助企业建立有效的软件工程实践。还是一家创业公司的创始人，专注于为特斯拉Model S系列电动车生产定制产品。



布鲁斯 R. 马克西姆（Bruce R. Maxim）曾任软件工程师、项目经理、教授、作家和咨询师，研究兴趣涉及软件工程、人机交互、游戏设计及教育等领域，曾任某游戏开发公司的首席技术官。现任密歇根大学迪尔伯恩分校副教授，为该校工程与计算机科学学院建立了游戏实验室。



Mc
Graw
Hill
Education

www.mheducation.com

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn



上架指导：计算机\软件工程

ISBN 978-7-111-54897-3



9 787111 548973 >

定价：99.00元